A Tensor-Based Formalization of the Event Calculus*

Efthimis Tsilionis[1,2], **Alexander Artikis**[3,2] and **Georgios Paliouras**[2]
[1]Department of Informatics & Telecommunications, National and Kapodistrian University of Athens, Greece
[2]Institute of Informatics & Telecommunications, NCSR "Demokritos", Greece
[3]Department of Maritime Studies, University of Piraeus, Greece
eftsilio@{di.uoa.gr, iit.demokritos.gr}, a.artikis@unipi.gr, paliourg@iit.demokritos.gr

Abstract

We present a formalization of the Event Calculus (EC) in tensor spaces. The motivation for a tensor-based predicate calculus comes from the area of composite event recognition (CER). As a CER engine, we adopt a logic programming implementation of EC with optimizations for continuous narrative assimilation on data streams. We show how to evaluate EC rules algebraically and solve a linear equation to compute the corresponding models. We demonstrate the scalability of our approach with the use of large datasets from a real-world application domain, and show it outperforms significantly symbolic EC, in terms of processing time.

1 Introduction

The Event Calculus (EC) is a first-order logical formalism for representing and reasoning about events and their effects [Kowalski and Sergot, 1986]. EC introduces the concept of *inertia*, which states that the effect of an event holds continuously in time if it is not disrupted by the effects of other events. Our work is motivated by the area of composite event recognition (CER). CER refers to the process of consuming a stream of time-stamped, simple derived events (SDEs), such as events coming from sensors, and identifying the time periods composite events (CE)s of interest — collections of events that satisfy a given pattern — hold. The definition of a CE imposes temporal and, possibly, atemporal constraints on its sub-events (SDEs or other CEs), and may be combined with static background knowledge [Giatrakos *et al.*, 2020].

Logic-based approaches have been applied to CER, since they exhibit a formal, declarative semantics, and at the same time support efficient reasoning [Dousson and Maigat, 2007; Cugola and Margara, 2010; Paschke and Bichler, 2008]. As a CER engine, we adopt a logic programming implementation of the EC [Skarlatidis *et al.*, 2015]. To reduce the complexity, we handle the input stream by means of windowing and employ caching techniques to avoid unnecessary recomputations [Artikis *et al.*, 2015].

Streaming environments, which are typical in CER, are characterized by the high rate and volume of input data. Therefore, the development of scalable reasoning techniques that can deal with large amounts of data is essential. One promising approach to scalable logical inference is the computation of program models using linear algebraic operations. Algebraic computation has been extensively studied and there are various libraries that provide efficient implementations of algebraic operations. Furthermore, the presence of parallel versions of these processing algorithms, as well as the recent advancements in hardware resources, e.g., GPUs, favor the employment of numerical computation for inference.

Along these lines, Sakama *et al.* [2021] represent propositional logic programs as matrices or tensors and through multiplication and a non-linear operation, they compute models of programs. An optimization with sparse matrices is introduced in [Nguyen *et al.*, 2022]. Sato [2017a] proposes a technique for obtaining the truth value of first-order logic formulas, where entities, logical connectives and existential quantifiers are formalized in tensor spaces. A procedure for query answering, where a query can be any nested formula, is also presented. This framework is applied to a Datalog program, where the transitive closure of a binary relation is computed by solving a system of linear matrix equations [Sato, 2017b].

We propose tensor-EC, a linear algebraic formulation of EC for CER under perfect model semantics. We map entities and time-points to vectors and events/fluents to matrices/tensors, depending on their arity. EC predicates increase by 1 the order of tensors by incorporating the temporal dimension. Our approach is not limited to square matrices or cubical tensors, since the entities and temporal sets can be of different size. Moreover, we show how to evaluate rules of an EC program, and solve a linear equation that produces the time-points at which a p-ary fluent holds. The resulting tensors, representing fluents, constitute the perfect model of the program. The contributions of this paper are:

- We present a translation of the language of EC in tensor spaces and show how to evaluate rules using algebraic operations.
- We evaluate experimentally tensor-EC on real data from the maritime domain, where we simulate a streaming environment and employ an EC program with many rules.
- We compare tensor-EC against the logic programming implementation of EC, and show that the former improves significantly reasoning time.

*https://cer.iit.demokritos.gr/publications/papers/2024/tensor-EC.pdf

Predicate	Meaning
happensAt$(e(X,Y),\ T)$	Event e for variables X and Y occurs at time T
holdsAt$(fl(X,Y){=}v,\ T)$	Fluent fl takes value v for variables X and Y at T
initiatedAt$(fl(X,Y){=}v,\ T)$	At T the fluent $fl(X,Y){=}v$ is initiated
terminatedAt$(fl(X,Y){=}v,\ T)$	At T the fluent $fl(X,Y){=}v$ is terminated

Table 1: Main predicates of Event Calculus (EC).

2 Background: Event Calculus

In this section, we present the EC dialect we adopt, as well as the logical inference procedure for CER.

2.1 Language

The time model of EC is linear and includes integer time-points [Skarlatidis *et al.*, 2015]. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. If fl is a fluent — a property that is allowed to have different values at different points in time — the term $fl(X,Y){=}v$ denotes that fluent fl has value v for variables X and Y. Boolean fluents are a special case in which the possible values are true and false. The predicate holdsAt$(fl(X,Y){=}v,\ T)$ is true if $fl(X,Y){=}v$ at time-point T. A fluent fl takes at most one value at each time-point. Event occurrences are expressed through the happensAt predicate. happensAt$(e(X,Y),\ T)$ denotes that event e occurs at time-point T for variables X and Y. Table 1 summarizes the available predicates of the EC language. EC events express instantaneous SDEs, while fluent-value pairs express SDEs and CEs that persist in time. Without loss of generality, we restrict our attention to events and fluents with arity ≤ 2.

The application-specific part of a formalization in EC is called *event description*.

Definition 1 (Event description)**.** *An event description comprises:*

(a) Ground happensAt *and* holdsAt *predicates. These are the facts and constitute the input (SDEs) to the system.*

(b) Rules with initiatedAt *and* terminatedAt *predicates at the head, expressing the effects of events on fluents.* ■

We focus on the task of computing the time-points for which a fluent has a particular value.

Definition 2 (Syntax)**.** initiatedAt *rules have the following syntax:*

$$\begin{aligned}&\text{initiatedAt}(fl(X,Y){=}v,\ T) \leftarrow\\ &\quad\text{happensAt}(e(X,Y),\ T),\\ &\quad[[\ [\text{not}]\ \text{happensAt}(a(X,Y),\ T),\ldots,\\ &\quad\ \ [\text{not}]\ \text{happensAt}(b(X,Y),\ T),\\ &\quad\ \ [\text{not}]\ \text{holdsAt}(c(X,Y){=}v_c,\ T),\ldots,\\ &\quad\ \ [\text{not}]\ \text{holdsAt}(d(X,Y){=}v_d,\ T).\ ]]\end{aligned} \tag{1}$$

Rule (1) *comprises conjunctions, meaning that all body literals should be satisfied in order for the rule to fire.* not *denotes negation by failure* [Clark, 1977]*, while* [not] *denotes that* 'not' *is optional. The variable T, present at the head and all body literals, constrains all literals to be evaluated at the same time-point. We use the term 'positive' to refer to events and fluents that must occur or hold at T, and the term 'negative' for events and fluents that should not occur or hold at T (symbol* not*). Rules of type* (1) *are Horn clauses and not restricted in the number of body literals. The only requirement is the first body literal to be a 'positive'* happensAt *predicate, which can then be followed by a possibly empty set of 'positive/negative'* happensAt *and* holdsAt *predicates, denoted by '[[ ]]'. Additionally, rules are 'safe', i.e. every variable that appears in the head of the rule or in any negative literal in the body also appears in at least one positive literal in the body.* terminatedAt *rules have a similar form.* ■

In Def. 2, we restrict the first body literal to be a positive happensAt predicate for complexity reasons. The time-points at which a fluent holds are usually a lot more than the time-points at which an event occurs. By selecting a positive happensAt predicate as the first body literal of initiatedAt and terminatedAt rules, we reduce complexity.

Example 1. *An example fluent definition from the maritime domain, is the following:*

$$\begin{aligned}&\text{initiatedAt}(gap(Vessel){=}farFromPorts,\ T) \leftarrow\\ &\quad\text{happensAt}(gap\_start(Vessel),\ T),\\ &\quad\text{not holdsAt}(nearPorts(Vessel){=}\text{true},\ T).\\ &\text{terminatedAt}(gap(Vessel){=}farFromPorts,\ T) \leftarrow\\ &\quad\text{happensAt}(gap\_end(Vessel),\ T).\end{aligned} \tag{2}$$

Rule-set (2) *formalizes the notion of a 'communication gap'* [Pitsikalis *et al.*, 2019]*. Communication gaps occur when a vessel is not emitting its position, either due to the absence of a nearby receiving station or on purpose. In maritime situational awareness, communication gaps may indicate an intention of hiding (e.g. in cases of illegal fishing). A gap is initiated for a Vessel if a gap_start has occurred far from ports, and terminated when a gap_end event is detected.*

The time-points produced by initiatedAt and terminatedAt rules are used to specify the time-points a fluent has a particular value. According to the *law of inertia*, a fluent holds continuously if it has been initiated and not terminated in the meantime. For example, if $fl(X,Y){=}v$ was initiated at T_s and terminated at T_f, with $T_s < T_f$, it holds for every time-point between T_s and T_f, excluding T_s, i.e., $T_{s+1}, T_{s+2}, \ldots, T_{f-1}, T_f$.

Definition 3 (*Inertia* axiom)**.** *The law of inertia is formalized by the following axiom:*

$$\begin{aligned}&\text{holdsAt}(fl(X,Y){=}v,\ T) \leftarrow\\ &\quad\text{initiatedAt}(fl(X,Y){=}v,\ T_{prev}),\\ &\quad\text{not terminatedAt}(fl(X,Y){=}v,\ T_{prev}),\\ &\quad next(T_{prev}, T).\\ &\text{holdsAt}(fl(X,Y){=}v,\ T) \leftarrow\\ &\quad\text{holdsAt}(fl(X,Y){=}v,\ T_{prev}),\\ &\quad\text{not terminatedAt}(fl(X,Y){=}v,\ T_{prev}),\\ &\quad next(T_{prev}, T).\end{aligned} \tag{3}$$

The *inertia* axiom (3) is a disjunction of two rules. The predicate $next(T_{prev}, T)$ present in both rules, denotes that the

next time-point after T_{prev} is time-point T. Notice that the *inertia* axiom allows reasoning about time-points that are not initiation or termination points.

2.2 Semantics and Operation

The EC language supports non-monotonic reasoning through negation-by-failure [Clark, 1977]. The *event description* (Def. 1) along with the *inertia* axiom constitute an EC program P. A common case in CER is the employment of hierarchical CE definitions [Giatrakos *et al.*, 2020]. Hierarchy in an EC program can be achieved through stratification [Apt *et al.*, 1988]. For instance, stratum P_0 may comprise all the events and those fluents that do not depend on events or other fluents, serving as input to the system (these are the ground happensAt and holdsAt facts, Def. 1). Fluents of stratum P_1 can be defined only in terms of events from P_0 while fluents of stratum $P_{s>1}$ can be defined in terms of at least one event from P_0, one fluent-value of P_{s-1}, and a possibly empty set of fluent-values from lower strata. The EC dialect that we use expresses locally stratified programs that may not necessarily be stratified [Przymusinski, 1987]. Note that local stratification is a standard assumption in EC [Artikis *et al.*, 2015].

As a CER engine, we adopt a logic programming implementation of EC [Skarlatidis *et al.*, 2015], equipped with optimization and caching techniques that make it suitable for continuous narrative assimilation on data streams [Artikis *et al.*, 2015]. Recognition is performed by processing hierarchical definitions in a bottom-up manner, whereby the ground events and fluents at the bottom of the hierarchy (stratum P_0) are processed first and all their time-points are cached. Subsequently, fluents of the next stratum P_1 are processed, their time-points are cached, and stratum-by-stratum the top of the hierarchy is reached. This way, when evaluating rules of stratum P_s, the time-points of the fluents and events of the body literals are fetched from the cache, avoiding unnecessary re-computations.

The CER process aims at the computation of all time-points at which CEs hold. This process takes place at specified query times $q_1, q_2, \ldots$. The recognition at each q_i is performed over the SDEs (input) that fall within a specified interval, the 'working memory' or window ω. All SDEs outside the window are discarded and not considered during recognition. This means that at each q_i CER depends only on the SDEs that took place in the interval $(q_i - \omega, q_i]$. This way, the cost of reasoning depends on the size of ω and not on the complete stream. The size of ω, as well as the temporal distance between two consecutive query times — the slide step $q_i - q_{i-1}$ — are user-specified.

In addition to events and fluent-value pairs, the domain of an application contains a nonempty set $\mathcal{C}$ of N constants $\{c_1, \ldots, c_N\}$, called domain entities. For example, in the fluent-value pair $gap(Vessel)$=$farFromPorts$ in rule (2), the variable $Vessel$ is mapped to vessel IDs. Additionally, there is a nonempty ordered set $\mathcal{T}$ of Ω constants $\{t_1, \ldots, t_\Omega\}$, that correspond to the time-points specified by the application's temporal window ω. The time variable T in the EC predicates is mapped to some time-point t_k, $q_i - \omega < t_k \leq q_i, \forall\, 1 \leq k \leq \Omega$. Recall from *inertia* axiom (3), the predicate $next(T_{prev}, T)$. In case $T_{prev} \rightarrow t_\Omega$, variable T cannot be mapped to a constant and thus predicate $next(t_\Omega, T)$ returns false. This way we restrict fluents to hold inside window ω.

The sets $\mathcal{C}$ and $\mathcal{T}$ (ground terms) constitute the Herbrand universe U_P of P, which is fixed and finite at each ω. If X is the set of all atoms of P, the Herbrand base (set of all ground atoms) of P is B_P=X^{U_P}. A model $\mathbf{M}_P$ of P is the set of ground atoms ($\subseteq B_P$) that makes all the rules of the program true. Since P is stratified, $\mathbf{M}_P$ is the unique perfect model of P [Gelfond and Lifschitz, 1988]. When a ground EC predicate r is entailed by $\mathbf{M}_P$, we write $\mathbf{M}_P \models r$.

The computation of $\mathbf{M}_P$ is performed at each query time q_i. Notice that, in the worst case, a fluent of a stratum may hold for the entire window ω, meaning that to compute all the time-points at which it holds we have to iterate through all the Ω constants in $\mathcal{T}$ (*inertia* axiom (3)).

3 Linear Algebraic Approach

We present our method, tensor-EC, for computing a model $\mathbf{M}_P$ of an EC program P in tensor spaces. Before we delve into the details of the approach, we provide terminology and notation used henceforth.

3.1 Preliminaries

Vectors are represented by bold lower case letters, e.g., $\mathbf{x}$. A vector of all ones is represented by $\mathbf{1}$. $\mathbf{x} \bullet \mathbf{y} = \mathbf{x}^\top \mathbf{y}$ is the dot product while $\mathbf{x} \circ \mathbf{y} = \mathbf{x}\mathbf{y}^\top$ is their outer product. Matrices are written by bold upper case letters like $\mathbf{X}$ and the identity matrix is denoted by $\mathbf{I}$. An order-p tensor (p specifies the number of dimensions, where $p > 2$) is written as $\underline{\mathbf{X}}$. $\underline{\mathbf{X}} \odot \underline{\mathbf{Y}}$ is the Hadamard product (element-wise multiplication) of two tensors and is defined only on two tensors of the same order and size. We refer to an element of a vector $\mathbf{x}$ or an order-p tensor $\underline{\mathbf{X}}$, as $\mathbf{x}_i$ and $\underline{\mathbf{X}}_{i_1,\ldots,i_p}$, respectively.

Definition 4 (mode-(n, m) product). *Let $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ be two order-p and order-k tensors, respectively. The mode-(n, m) contracted product $\underline{\mathbf{X}} \times_{n,m} \underline{\mathbf{Y}}$ of $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ is defined as:*

$$(\underline{\mathbf{X}} \times_{n,m} \underline{\mathbf{Y}})_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_p,j_1,\ldots,j_{m-1},j_{m+1},\ldots,j_k} = \sum_z \underline{\mathbf{X}}_{i_1,\ldots,i_{n-1},z,i_{n+1},\ldots,i_p} \underline{\mathbf{Y}}_{j_1,\ldots,j_{m-1},z,j_{m+1},\ldots,j_k} \quad . \quad \blacksquare$$

In Def. 4, (n, m) index the dimensions of the two operands. Then, each element of the resulting tensor is the dot product of the fibers of size $|z|$ of the n-th dimension of $\underline{\mathbf{X}}$ and the fibers of size $|z|$ of the m-th dimension of $\underline{\mathbf{Y}}$.

3.2 Encoding EC in Tensor Spaces

The EC language contains the sets of constants, $\mathcal{C}$ and $\mathcal{T}$ (section 2.2), events/fluents, and the predicates outlined in Table 1. We encode entities c_i from $\mathcal{C}$ in one-hot vectors $\mathbf{c}_i$, i.e., vectors that have one at the i-th position and zeros elsewhere. Similarly, we encode time-points t_k from $\mathcal{T}$ in one-hot vectors $\mathbf{t}_k$. The EC sets of constants now become $\mathcal{C}' = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$ and $\mathcal{T}' = \{\mathbf{t}_1, \ldots, \mathbf{t}_\Omega\}$, forming the standard basis of $\mathbb{R}^N$ and $\mathbb{R}^\Omega$, respectively. When it is not clear from the context, we will specify the size of a vector $\mathbf{x}$ with $\mathbf{x}^N$ or $\mathbf{x}^\Omega$.

EC predicates are translated into matrices or tensors. The shape/order of a matrix/tensor equals the arity of

events/fluents plus 1 for the temporal dimension. For illustration purposes, we restrict attention to binary events/fluents.

Definition 5 (EC predicates encoding)**.** *An EC predicate r is encoded by an order-3 tensor $\underline{\mathbf{R}} \in \{0,1\}^{N\times N\times\Omega}$, where:*

$$\underline{\mathbf{R}}_{i,j,k} = \begin{cases} 1, & \text{if } \mathbf{M}_P \models r, \text{ for } c_i, c_j, t_k \\ 0, & \text{o.w} \end{cases}$$
$$\forall\, 1 \leq i,j \leq N, 1 \leq k \leq \Omega \ . \qquad \blacksquare$$

Element $\underline{\mathbf{R}}_{i,j,k}$ equals 1 if predicate r is true in the model $\mathbf{M}_P$ of the program for variable groundings c_i, c_j, t_k, and 0 if it is not. The example below illustrates this encoding.

Example 2. *Assume that $\mathcal{C}$=$\{c_1, c_2\}$ has two entities, say vessel IDs, and $\mathcal{T}$=$\{t_1, t_2\}$ has two time-points. Then, $\mathcal{C}'$=$\{\mathbf{c}_1, \mathbf{c}_2\}$ and $\mathcal{T}'$=$\{\mathbf{t}_1, \mathbf{t}_2\}$. Furthermore, assume the following groundings of the EC predicate* $\mathsf{initiatedAt}(fl(X,Y){=}v,\ T)$*, expressing the initiation points of fluent fl:*

$$\mathsf{initiatedAt}(fl(c_1,c_2){=}v, t_1)$$
$$\mathsf{initiatedAt}(fl(c_1,c_2){=}v, t_2)$$
$$\mathsf{initiatedAt}(fl(c_2,c_1){=}v, t_2) \quad .$$

Below we present, the one-hot vector of $\mathbf{c}_1$ (left), the one-hot vector of $\mathbf{t}_2$ (middle), and the tensor $\underline{\mathbf{S}}$ encoding the EC predicate $\mathsf{initiatedAt}(fl(X,Y){=}v,\ T)$ *(right):*

$$\mathbf{c}_1 = \begin{bmatrix}1\\0\end{bmatrix},\ \mathbf{t}_2 = \begin{bmatrix}0\\1\end{bmatrix},\ \underline{\mathbf{S}} = \left[\begin{array}{cc|cc} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right]. \tag{4}$$

The vertical line in the above tensor representation serves the separation of the temporal dimension, i.e., it separates the two temporal slices. In this example, t_1 is expressed by the left slice while t_2 is expressed by the right slice. The rows and columns of the tensor correspond to entities c_1 and c_2. For example, the first row and column of each temporal slice refer to c_1. When we want to refer to a slice i of a tensor $\underline{\mathbf{S}}$, we use the notation $\underline{\mathbf{S}}_{:,:,i}$. In the tensor representation in (4)*, a value of 1 signifies that for specific groundings of the variables the predicate is true and a value of 0 that the predicate is false. For example, the ground predicate* $\mathsf{initiatedAt}(fl(c_1,c_2){=}v, t_2)$ *corresponds to the element $\underline{\mathbf{S}}_{1,2,2}$ of $\underline{\mathbf{S}}$ with value 1, and states that fluent fl is initiated at time-point t_2 for entities c_1 and c_2.*

To query the truth value of a specific variable grounding, e.g. $\mathsf{initiatedAt}(fl(c_i,c_j){=}v, t_k)$, we use the following:

$$\begin{aligned} &\mathsf{initiatedAt}(fl(c_i,c_j){=}v, t_k) = \\ &\underline{\mathbf{S}} \times_{1,1} \mathbf{c}_i \times_{2,1} \mathbf{c}_j \times_{3,1} \mathbf{t}_k = \\ &\underline{\mathbf{S}}_{i,j,k} \in \{0,1\}, \\ &\forall\, 1 \leq i,j \leq N, 1 \leq k \leq \Omega \ . \end{aligned} \tag{5}$$

3.3 Reasoning in Tensor-EC

The goal in EC is to compute the time-points at which a fluent holds. To achieve this, we first need to evaluate initiation and termination rules. To do this algebraically, we next show how we treat negation, conjunction, and disjunction.

EC predicates that participate negatively in the body of a rule (symbol not in rule (1)), imply that an event or fluent should not occur or hold at a specific time-point. To obtain a tensor representing a negative literal, we subtract from 1 each element of the tensor encoding the corresponding positive literal. Consider the negative literal $\mathsf{not\ happensAt}(a(X,Y),T)$. The tensor $\neg\underline{\mathbf{A}}$ used to represent this negative literal is computed as per Def. 6.

Definition 6 (Tensor Negation)**.** *Negation is defined as:*

$$\neg\underline{\mathbf{A}} = \mathbf{1}^N \circ \mathbf{1}^N \circ \mathbf{1}^\Omega - \underline{\mathbf{A}} \ \in \{0,1\}^{N\times N\times\Omega}. \qquad \blacksquare$$

In Def. 6, notice that the outer product of all-ones vectors results in an all-ones order-3 tensor. $\neg\underline{\mathbf{A}}$ is the result of subtracting from 1 all the elements of the positive counterpart tensor, i.e., $\underline{\mathbf{A}}$.

Example 3. *The negation of $\underline{\mathbf{S}}$ from* (4) *is:*

$$\neg\underline{\mathbf{S}} = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right].$$

Multiplication is used to evaluate the conjunction of literals. In EC rules conjunctive literals are evaluated at the same time-point, and usually on the same entities. Consider the following conjunction:

$$\mathsf{happensAt}(a(X,Y),T),\ \mathsf{holdsAt}(b(X,Y),T)\ .$$

We denote each predicate with tensors $\underline{\mathbf{A}}$ and $\underline{\mathbf{B}}$, respectively, and define tensor conjunction as per Def. 7.

Definition 7 (Tensor Conjunction)**.** *Conjunction is defined as the Hadamard product of two tensors:*

$$\underline{\mathbf{A}} \odot \underline{\mathbf{B}} \quad . \qquad \blacksquare$$

Example 4. *Consider the following tensors $\underline{\mathbf{A}}$ and $\underline{\mathbf{B}}$:*

$$\underline{\mathbf{A}} = \left[\begin{array}{cc|cc} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{array}\right],\ \underline{\mathbf{B}} = \left[\begin{array}{cc|cc} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right].$$

Their conjunction would be:

$$\left[\begin{array}{cc|cc} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right],$$

stating that event a and fluent b have the same grounding of variables for entities c_1 and c_2 only at time-point t_2.

In the Technical Appendix, we present a conjunction of literals in the presence of an existentially quantified variable. In that case, apart from multiplying the dimensions specified by the variables that are common among the literals, we also sum the result across the dimension indexed by the quantified variable.

Disjunction is treated by tensor addition. Consider that we want to compute the following disjunction of literals:

$$\mathsf{happensAt}(a(X,Y),T) \vee \mathsf{holdsAt}(b(X,Y),T)\ .$$

Definition 8 (Tensor Disjunction)**.** *Disjunction is defined as:*

$$\underline{\mathbf{A}} + \underline{\mathbf{B}} \ \in \mathbb{R}^{N\times N\times\Omega} \xrightarrow{\theta_{>1}} \{0,1\}^{N\times N\times\Omega}. \qquad \blacksquare$$

Notice that in case of tensor disjunction, the value of an element of the resulting tensor may be greater than 1. In this case, we employ a standard thresholding operation, denoted by $\xrightarrow{\theta_{>1}}$, that makes every entry greater than 1 equal to 1. The same applies to disjunctive rules (rules with the same head), as well as for conjunction in the presence of existentially quantified variables (see Technical Appendix). This operation is needed because a tensor may participate negatively in the body of a rule and the computation of its negation will result in negative values, if thresholding is not used.

3.4 Computing a Model in Tensor Spaces

To obtain the time-points a fluent holds, we need to compute its initiation and termination points. In rule (1), we presented the general syntax of initiatedAt and terminatedAt rules. Let $\underline{\mathbf{S}}$ be the tensor encoding the initiation points of fluent $fl(X,Y)=v$, $\underline{\mathbf{E}}$, $\underline{\mathbf{A}}$, $\underline{\mathbf{B}}$ the tensors encoding the event occurrences of events $e(X,Y)$, $a(X,Y)$, and $b(X,Y)$, respectively, and $\underline{\mathbf{C}}$, $\underline{\mathbf{D}}$ the tensors encoding the time-points at which fluents $c(X,Y)=v_c$ and $d(X,Y)=v_d$ hold, where $\underline{\mathbf{S}}, \underline{\mathbf{E}}, \underline{\mathbf{A}}, \underline{\mathbf{B}}, \underline{\mathbf{C}}, \underline{\mathbf{D}} \in \{0,1\}^{N\times N\times\Omega}$. Recall from rule (1) that the symbol [not] denotes that negation is optional. If a literal participates negatively in the body of a rule, the tensor encoding the predicate is negated according to Def. 6. To compute the initiation points of $\underline{\mathbf{S}}$, we have:

$$\underline{\mathbf{S}} = \underline{\mathbf{E}} \odot \Big[\Big[\, {}^{[\neg]}\underline{\mathbf{A}} \odot {}^{[\neg]}\underline{\mathbf{B}} \odot {}^{[\neg]}\underline{\mathbf{C}} \odot {}^{[\neg]}\underline{\mathbf{D}}\Big]\Big] \tag{6}$$

The symbol $^{[\neg]}$ represents the optional negation of tensors. '$\big[\big[\ \big]\big]$', similarly to rule (1), denotes that the presence of the enclosed tensors is optional. The evaluation of rule (1) in tensor spaces, is the Hadamard product of the tensors encoding the body literals, and the result is tensor $\underline{\mathbf{S}}$ encoding the initiation points of fluent fl. In Eq. (6), the initiation points of $\underline{\mathbf{S}}$ are basically the time-points at which all tensors, $\underline{\mathbf{E}}$, $\big[\big[\, {}^{[\neg]}\underline{\mathbf{A}}$, ${}^{[\neg]}\underline{\mathbf{B}}$, ${}^{[\neg]}\underline{\mathbf{C}}$, ${}^{[\neg]}\underline{\mathbf{D}}\big]\big]$, are true for the same groundings of entities and time-points.

terminatedAt rules are evaluated in a similar manner. The reasoning process ends with the computation of the time-points at which a fluent holds. To achieve this in tensor-EC, we employ the *inertia* axiom presented in rule-set (3). The *inertia* axiom is a disjunction of rules, where the variables are common among the first two body literals of each rule. Let $\underline{\mathbf{S}}, \underline{\mathbf{T}} \in \{0,1\}^{N\times N\times\Omega}$ be order-3 tensors, computed by the equivalent tensor formulation (Eq. (6)) of rules of type (1), that encode the time-points at which fluent fl is initiated and terminated, respectively. $\neg\underline{\mathbf{T}} \in \{0,1\}^{N\times N\times\Omega}$ is the negative version of $\underline{\mathbf{T}}$, computed as per Def. 6.

The predicate $next(T_{prev}, T)$ in *inertia* axiom (3) states that the next time-point of T_{prev} is T. We encode $next$ with shift matrix $\mathbf{U} \in \{0,1\}^{\Omega\times\Omega}$, that is, a square matrix with ones only on the super-diagonal and zeros elsewhere. Post-multiplying a matrix $\mathbf{A}$ with $\mathbf{U}$, i.e., $\mathbf{AU}$, results in shifting the elements of $\mathbf{A}$ to the right by one position, with zeros appearing in the first column. Multiplying a tensor $\underline{\mathbf{A}}$ with a shift matrix results in the shifting of elements along the temporal dimension, i.e., the first temporal slice is a matrix of zeros, $\mathbf{A}_{:,:,1} = \mathbf{0}^N \circ \mathbf{0}^N$, where $\mathbf{0}^N$ an all-zeros vector. Finally, let $\underline{\mathbf{H}}$ be the tensor encoding the time-points at which fluent fl holds. Then, the *inertia* axiom can be seen as a first-order difference (recursive) equation of the form:

$$\begin{aligned} &\underline{\mathbf{H}} = \big(\underline{\mathbf{S}} \odot \neg\underline{\mathbf{T}}\big) \times_{3,1} \mathbf{U} + \big(\underline{\mathbf{H}} \odot \neg\underline{\mathbf{T}}\big) \times_{3,1} \mathbf{U} \Leftrightarrow \\ &\underline{\mathbf{H}} - \big(\underline{\mathbf{H}} \odot \neg\underline{\mathbf{T}}\big) \times_{3,1} \mathbf{U} = \big(\underline{\mathbf{S}} \odot \neg\underline{\mathbf{T}}\big) \times_{3,1} \mathbf{U}\,. \end{aligned} \tag{7}$$

Eq. (7) states that a fluent holds at a time-point if it was initiated or held and not terminated at the previous time-point.

Unfolding Eq. (7) for every element of tensors $\underline{\mathbf{H}}, \underline{\mathbf{S}}, \neg\underline{\mathbf{T}}$, i.e., for every pair of entities $c_i, c_j \in \mathcal{C}$ and time-point $t_k \in \mathcal{T}$, we result in the following system of linear first-order difference equations:

$$\begin{aligned} \underline{\mathbf{H}}_{1,1,1} &= 0 \\ -\underline{\mathbf{H}}_{1,1,1}\neg\underline{\mathbf{T}}_{1,1,1} + \underline{\mathbf{H}}_{1,1,2} &= \underline{\mathbf{S}}_{1,1,1}\neg\underline{\mathbf{T}}_{1,1,1} \\ &\vdots \\ -\underline{\mathbf{H}}_{N,N,\Omega-1}\neg\underline{\mathbf{T}}_{N,N,\Omega-1} + \underline{\mathbf{H}}_{N,N,\Omega} &= \underline{\mathbf{S}}_{N,N,\Omega-1}\neg\underline{\mathbf{T}}_{N,N,\Omega-1}\,. \end{aligned}$$

Notice that the initial condition $\underline{\mathbf{H}}_{i,j,1}= 0,\ 1 \le i,j \le N$, states that at the first time-point t_1, regardless the pair of entities c_i, c_j, no fluent can hold. The above system can be written in matrix form as:

$$\underbrace{\begin{bmatrix} 1 & & & \\ -(\neg\underline{\mathbf{T}}_{1,1,1}) & 1 & & \\ & \ddots & \ddots & \\ & & -(\neg\underline{\mathbf{T}}_{N,N,\Omega-1}) & 1 \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} \underline{\mathbf{H}}_{1,1,1} \\ \underline{\mathbf{H}}_{1,1,2} \\ \vdots \\ \underline{\mathbf{H}}_{N,N,\Omega} \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} 0 \\ \underline{\mathbf{S}}_{1,1,1}\neg\underline{\mathbf{T}}_{1,1,1} \\ \vdots \\ \underline{\mathbf{S}}_{N,N,\Omega-1}\neg\underline{\mathbf{T}}_{N,N,\Omega-1} \end{bmatrix}}_{\mathbf{b}}, \tag{8}$$

where $\mathbf{G} \in \mathbb{R}^{N^2\Omega\times N^2\Omega}$ is the coefficients matrix, and $\mathbf{h} \in \mathbb{R}^{N^2\Omega}$, $\mathbf{b} \in \{0,1\}^{N^2\Omega}$, are column vectors. Our goal is to solve Eq. (8) for $\mathbf{h}$, i.e., the time-points at which a fluent holds for every pair of entities.

To construct the matrix equation (8), we must first perform a series of operations. We define $vec[\cdot]$ as the vectorization operator, which transforms a tensor into a vector. For example, let $\mathbf{a}$ be a vector and $\underline{\mathbf{A}}$ a tensor, $vec[\underline{\mathbf{A}}] : \underline{\mathbf{A}} \in \mathbb{R}^{N\times N\times\Omega} \to \mathbf{a} \in \mathbb{R}^{N^2\Omega}$. Then, the operations to produce $\mathbf{G}$ and $\mathbf{b}$ in Eq. (8), are the following:

(a) $\mathbf{G} \in \mathbb{R}^{N^2\Omega\times N^2\Omega}:\ \mathbf{G}_{i,i}= 1\,, \mathbf{G}^{\star}=-vec\big[\neg\underline{\mathbf{T}}\big]$, $\mathbf{G}_{i,j}= 0,\ \forall\, i,j : i \neq j, j \neq i-1$

(b) $\mathbf{b} = vec\Big[\Big(\big(\underline{\mathbf{S}} \odot \neg\underline{\mathbf{T}}\big) \times_{3,1} \mathbf{U}\Big)\Big] \in \{0,1\}^{N^2\Omega}$

In (a), all the elements of the principal diagonal of $\mathbf{G}$ are set to 1, the first sub-diagonal ($\mathbf{G}^{\star}$) is set to the result of vectorizing $\neg\underline{\mathbf{T}}$ multiplied by -1, and all the remaining elements are set to 0. Notice that, due to multiplication of $vec\big[\neg\underline{\mathbf{T}}\big]$ by -1, $\mathbf{G} \notin \{0,1\}^{N^2\Omega\times N^2\Omega}$ but $\mathbf{G} \in \{-1,0,1\}^{N^2\Omega\times N^2\Omega}$. Vector $\mathbf{b}$, in (b), is the vectorization of the Hadamard product of initiation and non-termination tensors, $\underline{\mathbf{S}}$ and $\neg\underline{\mathbf{T}}$, shifted (mode-(3,1) product) by matrix $\mathbf{U}$.

$\mathbf{G}$ is a lower unitriangular matrix, i.e., a lower triangular matrix for which all elements on the principal diagonal equal to 1. Additionally, $\mathbf{G}$ is a bi-diagonal matrix [Demmel, 1997; Kılıç and Stanica, 2013], since only the elements of the principal and the first sub-diagonal may differ from 0. Since $\mathbf{G}$ is unitriangular, its determinant is 1 (product of the principal diagonal elements), and thus, it has an inverse ($\mathbf{G}^{-1}$). Hence, Eq. (8) has a unique solution, that can be expressed formally for vector $\mathbf{h}$ and tensor $\underline{\mathbf{H}}$, both encoding the time-points at which $fl(X,Y)=v$ holds, as per Def. 9.

Definition 9 (*Inertia* Axiom Solution)**.** *The time-points at which a fluent-value pair holds are computed by:*

$$\begin{aligned} &\mathbf{h} = \mathbf{G}^{-1}\mathbf{b},\ \ \mathbf{h} \in \mathbb{R}^{N^2\Omega} \xrightarrow{\theta_{>1}} \{0,1\}^{N^2\Omega}, \\ &\underline{\mathbf{H}} = vec^{-1}\big[\mathbf{h}\big],\ \ \underline{\mathbf{H}} \in \{0,1\}^{N\times N\times\Omega} \quad . \end{aligned} \tag{9}$$

■

In Def. 9, $vec^{-1}\left[\ \cdot\ \right]$ is the inverse of the vectorization operator, i.e., transforms a vector to a tensor. This operation is needed, since $\underline{\mathbf{H}}$ may participate in the body of initiatedAt/terminatedAt rules of type (1) at higher strata. Notice, also, that $\mathbf{h} \in \mathbb{R}^{N^2\Omega}$. To constrain $\mathbf{h} \in \{0,1\}^{N^2\Omega}$ we use the thresholding operation $\xrightarrow{\theta_{>1}}$, already discussed in Section 3.3. Def. 9 may be extended for tensors of any order, i.e., for fluents with arity > 2.

The process described so far is repeated for every stratum of the EC program P, as is also the case for symbolic-EC (see Section 2.2). The tensors of stratum P_s, encoding the time-points at which fluents of P_s hold (computed as per Def. 9), are cached and propagated to higher strata ($P_{>s}$). At the end, the tensors of all strata constitute the perfect model $\mathbf{M}_P$.

Proposition 1 (Correctness)**.** *The unique solution of Eq.* (8)*, computed by Eq.* (9)*, coincides with the time-points at which a fluent-value pair holds, as expressed by the perfect model of the corresponding program in symbolic-EC.* ♦

The proof may be found in the Technical Appendix.

Proposition 2 (Complexity)**.** *The time complexity of solving Eq.* (8) *is* $\mathcal{O}(N^{p-1}\Omega)$ *for order-*p *tensors* [Demmel, 1997]. ♦

Eq. (8) requires the construction of the coefficients matrix $\mathbf{G}$ and vector $\mathbf{b}$. The first sub-diagonal of $\mathbf{G}$ depends on the non-termination tensor $\neg\underline{\mathbf{T}}$, while $\mathbf{b}$ depends on the Hadamard product of initiation and non-termination tensors, $\underline{\mathbf{S}}$ and $\neg\underline{\mathbf{T}}$, shifted (mode-(3,1) product) by matrix $\mathbf{U}$. $\underline{\mathbf{S}}$ and $\underline{\mathbf{T}}$ are produced by evaluating initiatedAt and terminatedAt rules of type (1) in tensor-EC (Eq. (6)), and require $\mathcal{O}(N^{p-1}\Omega)$ time for order-p tensors. The time complexity of the mode-(3,1) product $\left(\underline{\mathbf{S}} \odot \neg\underline{\mathbf{T}}\right) \times_{3,1} \mathbf{U}$, is $\mathcal{O}(N^{p-1}\Omega^2)$.

Evaluating axiom (3) in symbolic-EC for fluents with arity $p-1$, requires in the worst case $\mathcal{O}(N^{p-1}\Omega)$. Moreover, rules of type (1), in symbolic-EC, are bound by $\mathcal{O}(N^{p-1}\Omega^2)$ [Tsilionis *et al.*, 2022]. Hence, both methods, symbolic-EC and tensor-EC, are bound theoretically by the same complexity.

However, the performance of tensor-EC can be boosted through parallelism or/and the employment of sparse representations. In this paper, we do not exploit parallelism (it is left for future work), but note that operations such as the Hadamard and mode-(n,m) products are trivially parallelized. In real-life scenarios, the time-points at which fluents are initiated and terminated are usually very few, and thus, the corresponding tensors would be very sparse. Operations on sparse representations avoid unnecessary calculations by not examining null elements (time-points at which the EC predicates are false), resulting in peformance improvement [Nguyen *et al.*, 2022]. Furthermore, recall that shift matrix $\mathbf{U}$ has 1s only on the first super-diagonal and its sparse structure can also be considered. Finally, matrix $\mathbf{G}$ in Eq. (8) is a bi-diagonal matrix, where only the elements of the principal and the first sub-diagonal may differ from 0. By taking advantage of the sparsity of $\mathbf{G}$ and the fact that the elements of the principal diagonal are equal to 1, the time of solving Eq. (8) can be further reduced (see in the Technical Appendix the computation of the inverse $\mathbf{G}^{-1}$). In the empirical analysis, we employ sparse representations for the tensors and matrices needed by tensor-EC and observe significant improvements.

4 Empirical Analysis

We present an empirical analysis on real datasets from the field of maritime monitoring.

4.1 Experimental Setup

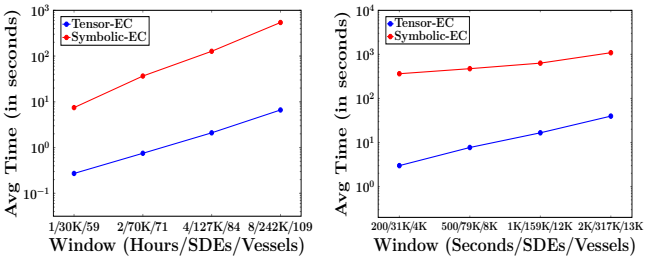
Symbolic-EC is implemented in XSB Prolog while the tensor-based implementation is written in Python. The source code of both methods and a subset of one of the datasets, are available in the Code & Data Appendix. The experiments were performed on a single core, on a computer with AMD EPYC 7543 and 400 GB of RAM, running Debian GNU/Linux 12, XSB Prolog 5.0.0 and Python 3.11.4.

The composite event recognition (CER) process involves the computation and caching of all time-points at which fluent-value pairs, expressing CEs, hold. On the field of maritime monitoring, CER concerns the recognition of composite maritime events (recall 'communication gap' from (2)) and is typically achieved by monitoring the messages vessels emit while sailing at sea. These messages are exchanged through the Automatic Identification System (AIS) [Bereta *et al.*, 2021] and contain information about the position, heading, speed, etc. of vessels at different points in time. Moreover, these messages can be annotated automatically, conveying information about the start/end of sailing at a low/high speed, changes in speed/heading, entrance or exit in an area of interest, etc. [Patroumpas *et al.*, 2017]. The annotated AIS messages constitute the input to our system.

The CE description used in our empirical analysis includes forty input events and twenty two fluents. Recall that each fluent is defined by one or more initiatedAt and terminatedAt rules, plus the *inertia* axiom. We employed two datasets for our empirical analysis; the first is a publicly available dataset, concerning approx. $5K$ vessels sailing in the Atlantic Ocean around the port of Brest, France, and consists of approx. $15M$ SDEs. The second dataset is proprietary and was provided to us by IMIS Global. It concerns $34K$ vessels sailing in the European seas and consists of approx. $32M$ SDEs. These datasets allow us to test the scalability of the methods.

Recall from Section 2.2, that CER takes place at specified query times $q_1, q_2, \ldots$, where the recognition at each q_i is performed over the SDEs (input — ground happensAt/holdsAt predicates) that fall within a user-specified window ω. To simulate a streaming behavior, the datasets are stored in CSV files and are processed periodically in chunks according to the window ω specification. Moreover, the slide step (distance between consecutive query times) is set equal to ω in the experiments, i.e., non-overlapping windows are used. Notice that, given a constant window ω, the number of SDEs varies from window to window and consequently, the number of vessels (the N constants of the entity set $\mathcal{C}$) changes, while the Ω time-points of set $\mathcal{T}$ (size of ω) remain unchanged.

Since the data concerning each vessel is very sparse due to periodic message emission, we use sparse representation of the tensors encoding fluents. However, efficient implementations of operations, such as the Hadamard or the mode-(n,m) product, on sparse tensors do not exist. Therefore, we convert tensors to matrices. For example an order-3 tensor $\underline{\mathbf{F}} \in \{0,1\}^{N\times N\times\Omega}$ is converted to a matrix $\mathbf{F} \in \{0,1\}^{N^2\times\Omega}$.



(a) Brest dataset (b) European seas dataset

Figure 1: Tensor vs. symbolic-EC.

4.2 Experimental Results

Figures 1(a) and (b) display our experimental results for the Brest and the European seas dataset, respectively. In both datasets, we employ temporal windows of four different sizes and on the x-axis we state the size of the window, as well as the average number of SDEs and average number of vessels/entities falling inside the window. The y-axis of both figures corresponds to average recognition time (in log scale).

In the Brest dataset the temporal window varies from 1 to 8 hours, while in the European seas dataset the window varies from 200 to 2000 seconds. In both datasets, the time-points of the windows correspond to seconds. Notice that the number of SDEs and vessels increases dramatically in the European seas dataset, even for smaller windows. Tensor-EC achieves a performance gain for all window sizes in both datasets, highlighting the efficiency of the method in comparison to the symbolic one. Recall that the experiments were performed on a single core, i.e., no parallelization was used.

5 Related Work

Computing models of logic programs in vector spaces has recently gained a lot of attention. Sakama *et al.* [2021] presented a method for computing models, where they encode propositional programs into matrices. Optimization techniques that make the method able to cope with huge programs in vector spaces, have also been proposed [Nguyen *et al.*, 2018; Nguyen *et al.*, 2021]. Our work is not directly comparable to these propositional approaches. We also compute the model of a program but we use a first-order language. Propositionalization in CER would lead to millions of propositional atoms in the Herbrand base, and the construction of the program matrix would be practically infeasible, due to huge memory requirements and time complexity. Additionally, the matrix construction process needs to be repeated at each temporal window, since the set of domain entities/vessels $\mathcal{C}$ does not remain the same in each window.

The work closest to ours is that by Sato [2017a; 2017b]. Sato [2017a] utilizes a first-order language with a finite domain of constants. Entities are represented by one-hot vectors and p-ary relations by order-p (cubical) tensors. Quantifiers are also encoded as tensors and their order depends on the number of appearances of the quantified variable. Then, a procedure, consisting of matrix multiplications or tensor mode-(n, m) products, is proposed for determining the truth value of nested quantified logic formulas. This procedure can also be used for computing the encoding of a rule head, but it is only applicable to square matrices encoding binary predicates. In the case of tensors, redundant computations are introduced, increasing the order of the resulting tensor (in the Technical Appendix we demonstrate the reason). To solve this issue, when the variables are common among the head and the body literals, as in a rule of type (1), we employ the Hadamard product in Eq. (6), a cheaper operation compared to matrix multiplication and mode-(n, m) products. A further improvement is the use of an evaluation approach that does not increase the order of the head tensor, in the presence of existentially quantified variables (see Technical Appendix).

The method for computing square matrices that constitute the least model of a transitive closure program in Datalog, is described in [Sato, 2017b]. The model is determined by solving a linear recursive equation and significant speedups, compared to symbolic systems, are observed. Our work is inspired by this study and in Eq. (7) we formalize the *inertia* axiom (3) as a linear recursive equation. Furthermore, we provide in Def. 9 a solution that can be extended to tensors of any order, as opposed to the square (binary predicates) matrix solution in [Sato, 2017b]. Consider again, the predicate $\mathsf{holdsAt}(fl(X,Y){=}v,\ T)$ that is encoded by tensor $\underline{\mathbf{H}} \in \{0,1\}^{N\times N\times\Omega}$. The corresponding square matrix would be $\mathbf{H} \in \{0,1\}^{N\Omega\times N\Omega}$. Expressing the *inertia* axiom as a discrete Sylvester equation, as proposed by Sato, and solving it would require $\mathcal{O}(N^3\Omega^3)$ time. In our approach, with the use of a bi-diagonal matrix, the time complexity is $\mathcal{O}(N^2\Omega)$, orders of magnitude lower. Recall that N in our empirical analysis reached a value of $13K$ in the European seas dataset.

Several EC implementations for logical reasoning over traces of events have been proposed in the literature [Chittaro and Montanari, 1996; Chesani *et al.*, 2010; Bragaglia *et al.*, 2012; Chesani *et al.*, 2013; Montali *et al.*, 2014; Arias *et al.*, 2022]. These approaches represent the whole history and thus, as the trace grows, they are unable to scale to streaming applications. Tensor-EC adopts the optimization techniques of [Artikis *et al.*, 2015], such as windowing and caching in hierarchical EC programs, to avoid unnecessary re-computations and scale to data streams.

6 Summary and Future Work

We proposed a linear algebraic approach for computing the perfect model of a hierarchical EC program. We represent EC predicates as tensors and demonstrate that the time-points any p-ary fluent (CE) holds can be assessed by solving a linear recursive equation. The scalability of our system is empirically demonstrated on real-world streaming data from the maritime domain. Additionally, our numerical approach improves the performance of the symbolical implementation of EC, by orders of magnitude. An interesting future work direction would be to develop tensor representations that avoid the grounding of every time-point at which a fluent holds. Encoding in tensors the time intervals a fluent holds continuously, will reduce substantially the inference time. Finally, we intend to exploit parallel algorithms of linear algebra and hardware resources (e.g., GPUs) to further boost performance.

Acknowledgements

This work was supported by the ENEXA "Efficient Explainable Learning on Knowledge Graphs" project (No 101070305), which has received funding from the EU Horizon Europe research and innovation programme.

References

[Apt *et al.*, 1988] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming.*, 1988.

[Arias *et al.*, 2022] Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. Modeling and reasoning in event calculus using goal-directed constraint answer set programming. *Theory Pract. Log. Program.*, 22(1):51–80, 2022.

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

[Bereta *et al.*, 2021] Konstantina Bereta, Konstantinos Chatzikokolakis, and Dimitris Zissis. Maritime reporting systems. In Alexander Artikis and Dimitris Zissis, editors, *Guide to Maritime Informatics*, pages 3–30. Springer, 2021.

[Bragaglia *et al.*, 2012] Stefano Bragaglia, Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. *Reactive Event Calculus for Monitoring Global Computing Applications*, pages 123–146. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Chesani *et al.*, 2010] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. A logic-based, reactive calculus of events. *Fundam. Inf.*, 105(1–2):135–161, jan 2010.

[Chesani *et al.*, 2013] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Representing and monitoring social commitments using the event calculus. *Auton. Agents Multi Agent Syst.*, 27(1):85–130, 2013.

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

[Clark, 1977] Keith L. Clark. Negation as failure. In *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, pages 293–322, 1977.

[Cugola and Margara, 2010] Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS 2010, Cambridge, United Kingdom, July 12-15, 2010*, pages 50–61, 2010.

[Demmel, 1997] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, USA, 1997.

[Dousson and Maigat, 2007] Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 324–329, 2007.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

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

[Kowalski and Sergot, 1986] Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.

[Kılıç and Stanica, 2013] Emrah Kılıç and Pantelimon Stanica. The inverse of banded matrices. *Journal of Computational and Applied Mathematics*, 237(1):126–135, 2013.

[Montali *et al.*, 2014] Marco Montali, Fabrizio M. Maggi, Federico Chesani, Paola Mello, and Wil M. P. van der Aalst. Monitoring business constraints with the event calculus. *ACM Trans. Intell. Syst. Technol.*, 5(1), jan 2014.

[Nguyen *et al.*, 2018] Hien D. Nguyen, Chiaki Sakama, Taisuke Sato, and Katsumi Inoue. Computing logic programming semantics in linear algebra. In Manasawee Kaenampornpan, Rainer Malaka, Duc Dung Nguyen, and Nicolas Schwind, editors, *Multi-disciplinary Trends in Artificial Intelligence - 12th International Conference, MIWAI 2018, Hanoi, Vietnam, November 18-20, 2018, Proceedings*, volume 11248 of *Lecture Notes in Computer Science*, pages 32–48. Springer, 2018.

[Nguyen *et al.*, 2021] Hien D Nguyen, Chiaki Sakama, Taisuke Sato, and Katsumi Inoue. An efficient reasoning method on logic programming using partial evaluation in vector spaces. *Journal of Logic and Computation*, 31(5):1298–1316, 03 2021.

[Nguyen *et al.*, 2022] Tuan Quoc Nguyen, Katsumi Inoue, and Chiaki Sakama. Enhancing linear algebraic computation of logic programs using sparse representation. *New Gen. Comput.*, 40(1):225–254, apr 2022.

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

[Patroumpas *et al.*, 2017] Kostas Patroumpas, Elias Alevizos, Alexander Artikis, Marios Vodas, Nikos Pelekis, and Yannis Theodoridis. Online event recognition from moving vessel trajectories. *GeoInformatica*, 21(2):389–427, 2017.

[Pitsikalis *et al.*, 2019] Manolis Pitsikalis, Alexander Artikis, Richard Dreo, Cyril Ray, Elena Camossi, and Anne-

Laure Jousselme. Composite event recognition for maritime monitoring. In *Proceedings of the 13th ACM International Conference on Distributed and Event-Based Systems*, DEBS '19, page 163–174, New York, NY, USA, 2019. Association for Computing Machinery.

[Przymusinski, 1987] T. Przymusinski. On the declarate semantics of stratified deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan, 1987.

[Sakama *et al.*, 2021] Chiaki Sakama, Katsumi Inoue, and Taisuke Sato. Logic programming in tensor spaces. *Annals of Mathematics and Artificial Intelligence*, 89, 12 2021.

[Sato, 2017a] Taisuke Sato. Embedding tarskian semantics in vector spaces. In *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence, Saturday, February 4-9, 2017, San Francisco, California, USA*, volume WS-17 of *AAAI Technical Report*. AAAI Press, 2017.

[Sato, 2017b] Taisuke Sato. A linear algebraic approach to datalog evaluation. *Theory and Practice of Logic Programming*, 17(3):244–265, 2017.

[Skarlatidis *et al.*, 2015] Anastasios Skarlatidis, Georgios Paliouras, Alexander Artikis, and George A. Vouros. Probabilistic event calculus for event recognition. *ACM Trans. Comput. Logic*, 16(2), feb 2015.

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