A Tensor-Based Probabilistic Event Calculus

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

Abstract

Complex Event Recognition (CER) systems receive as input a stream of time-stamped events and identify situations of interest that satisfy a given pattern. Streaming environments are characterized by the high rate and volume of input data, and thus, scalability is of crucial importance. At the same time, noise and uncertainty are ubiquitous in temporal data, and not considering them, leads to erroneous detections. To confront these challenges, we present a tensor-based formalization of the Event Calculus (EC) for probabilistic inference, and demonstrate the scalability of our approach with the use of CER datasets from two real-world application domains. Moreover, we demonstrate the benefits of our approach, in terms of processing time, by comparing it against a probabilistic logic programming implementation of EC.

1 Introduction

Complex event recognition (CER) is query answering over high-velocity and high-volume data streams. Queries are called *complex events* (CEs) and answering/identifying them consists of finding the time periods at which they hold. An input stream is composed of time-stamped, simple derived events (SDEs), such as events coming from sensors. A CE definition expresses a pattern of SDEs and/or other CEs (forming a hierarchy), where temporal and, possibly atemporal constraints, are imposed and may be combined with static background knowledge. At the same time, to favor real-time decision making, CER systems must detect the CEs near to the actual time of their occurrence. Therefore, scalability is of critical importance.

Logic-based approaches, due to their formal, declarative semantics, allow the succinct definition of activities of interest and at the same time support efficient reasoning. The Event Calculus (EC) is a first-order logical temporal formalism for representing and reasoning about events and their effects. A key feature of EC is the axiomatization 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. Logic programming implementations of EC have been applied successfully in many domains of interest, such as human activity recognition and maritime situational awareness (Tsilionis, Artikis, and Paliouras 2022).

The need to cope with big data has also led to the development of algebraic methods for logical inference (Sato 2017a; Sato 2017b; Sakama, Inoue, and Sato 2021; Tsilionis, Artikis, and Paliouras 2024). Computing logic models with the use of linear algebraic operations takes advantage of numerical computation, which has been extensively studied and efficiently implemented. Furthermore, scalability is favored by parallel algorithms of linear algebra and hardware resources (e.g., GPUs).

However, the aforementioned techniques (symbolic or not) assume that the input data are not affected by any type of uncertainty. In CER, specifically, data uncertainty is a frequent phenomenon (e.g., due to a sensor malfunction) and not considering it, leads to the erroneous detection of CEs. To handle uncertainty, various probabilistic CER frameworks have been proposed (see (Alevizos et al. 2017) for a survey). Regarding EC, probabilistic CER implementations (Skarlatidis et al. 2015a; Artikis, Makris, and Paliouras 2021; Mantenoglou, Artikis, and Paliouras 2023) consume events that are assigned a probability value, called probabilistic facts, and compute the *success* probabilities — *weighted model count* — of CEs to hold. All these approaches are based on ProbLog (Fierens et al. 2015), which performs probabilistic reasoning after grounding the logic program expressing the EC specification, and applying knowledge compilation (Darwiche and Marquis 2002). In streaming environments, these approaches fail to scale.

We propose tensor-pEC, a linear algebraic formulation of EC for probabilistic CER. We adopt an EC dialect with two types of constants, i.e., entities and time-points, and map them to vectors. EC predicates, stating that an event/fluent occurs/holds at a time-point, are represented by matrices or tensors, according to the number of involved entities. Moreover, we show how to evaluate logical connectives using algebraic operations, and solve a linear equation that produces the weighted model count of a fluent holding at each time-point separately. The contributions of this paper are summarized below:

- We present tensor-pEC, i.e., a tensor-based formalization of EC for probabilistic reasoning and show its equivalence to the probabilistic logic programming counterpart.
- We present a theoretical evaluation of tensor-pEC and demonstrate experimentally its efficiency on real data from two application domains, where we simulate a streaming environment and employ EC programs with

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).

many rules.

- We compare our method against the symbolic approach, and show that the former improves reasoning time by orders of magnitude.

2 Background

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

2.1 Event Calculus

Representation The time model of EC is linear and includes integer time-points (Skarlatidis et al. 2015a). The EC dialect also includes events and fluents, i.e., properties that are allowed to have different values at different points in time. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. 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$(fl(X,Y){=}v,\ T)$ denotes that $fl(X,Y){=}v$ holds 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. EC events express the instantaneous input SDEs, while fluent-value pairs express durative input SDEs and the output CEs, which typically take place over intervals. Table 1 summarizes the main predicates of this EC dialect. 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) Domain-dependent rules with initiatedAt *and* terminatedAt *predicates at the head, expressing the effects of events on fluents.* ■

CEs are defined by means of at least one initiatedAt and one terminatedAt rule.

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

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

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*). The first body literal is a 'positive'* happensAt *predicate to express the effects of events on fluents, which can then be followed by a possibly empty set of 'positive/negative'* happensAt *and* holdsAt *predicates, denoted by '$\big[\big[\ \big]\big]$'. Rules of type* (1) *are not restricted in the number of body literals and the only requirement is that the variables appearing in the head, must also appear in the body literals. In other words, we do not allow existential quantification.* terminatedAt *rules have a similar form.* ■

Example 1. *An example fluent definition from the field of human activity recognition, is the following:*

$$\begin{aligned}&\text{initiatedAt}(moving(P_1,P_2){=}\text{true},\ T) \leftarrow \\ &\quad \text{happensAt}(walking(P_1),\ T), \\ &\quad \text{happensAt}(walking(P_2),\ T), \\ &\quad \text{holdsAt}(close(P_1,P_2){=}\text{true},\ T), \\ &\quad \text{holdsAt}(similarOrientation(P_1,P_2){=}\text{true},\ T). \\ &\text{terminatedAt}(moving(P_1,P_2){=}\text{true},\ T) \leftarrow \\ &\quad \text{happensAt}(walking(P_1),\ T), \\ &\quad \text{holdsAt}(close(P_1,P_2){=}\text{false},\ T).\end{aligned} \tag{2}$$

Rule-set (2) *detects when two people, P_1 and P_2, start and stop moving together. The first rule in* (2) *states that moving is initiated if P_1 and P_2 are walking close to each other and have a similar orientation. The second rule states that moving is terminated when P_1 and P_2 start walking away from each other. The complete definition includes more* terminatedAt *rules, but they are not presented here to make the presentation easier to follow.*

The time-points produced by initiatedAt and terminatedAt rules are used to specify the time-points a fluent has a particular value. EC provides an axiomatization of the commonsense *law of inertia*, according to which, a fluent-value pair holds continuously if it has been initiated in the past and not 'broken' in the meantime. 'Broken' means that the fluent has been terminated or initiated with a different value. For example, if $fl(X,Y){=}v$ was initiated at T_s, broken at T_f but not broken earlier, with $T_s < T_f$, $fl(X,Y){=}v$ 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:*

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

■

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. The auxiliary predicate $\mathsf{broken}(fl(X,Y){=}v, T_{prev})$ checks whether fluent $fl(X,Y){=}v$ is terminated or $fl(X,Y){=}v'$ is initiated with a v' other than v.

Reasoning In CER, the task is to compute the time-points for which a fluent, representing a CE, has a particular value. This takes place at specified query times $q_1, q_2, \ldots$, where 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 $(q_i - \omega, q_i]$ and not on the complete stream. In order to find the time-points at which a CE holds, we evaluate, for each time-point included in $(q_i - \omega, q_i]$, the *inertia* axiom (3). During the evaluation, variables X, Y and T are ground. We repeat this evaluation for each fluent-value pair of the event description.

The grounding of the variables is carried out based on two sets of domain constants. The first set $\mathcal{C}{=}\{c_1, \ldots, c_N\}$, contains N constants, called domain entities. For example, in the fluent-value pair $moving(P_1, P_2){=}\mathsf{true}$ in rule (2), the variables P_1 and P_2 are mapped to person IDs. The second set $\mathcal{T}{=}\{t_1, \ldots, t_\Omega\}$ is ordered and contains Ω constants, corresponding to the time-points specified by the application's temporal window ω. The time variable T in the holdsAt predicate in the head of *inertia* axiom (3) is mapped to some time-point $t_k \in \mathcal{T}$, $q_i - \omega < t_k \leq q_i, \forall\ k : 1 \leq k \leq \Omega$. Recall from *inertia* axiom (3) the predicate $next(T_{prev}, T)$. In case $T \rightarrow t_1$ (the first time-point of window ω), T_{prev} cannot be mapped to a constant and thus predicate $next(T_{prev}, t_1)$ returns false. This way fluents are restricted to hold inside window ω.

2.2 Probabilistic Event Calculus

The EC dialect has been expressed in the probabilistic logic programming framework ProbLog (Fierens et al. 2015), and it is called Prob-EC after (Skarlatidis et al. 2015a). ProbLog employs the *possible world* semantics, where each world is constructed by including or not *probabilistic facts*. A probabilistic fact is denoted by $p :: f$, and indicates that (for specific groundings of the variables) atom f is true in a world with probability $p \in [0, 1]$. If f is not included in a world, then $\neg f$ is included with probability $1 - p$. Probabilistic facts are considered as independent random Boolean variables. Hence, the product of the probabilities of the facts (positive or negative) that constitute a world, is the probability of the world. Non-probabilistic facts are part of every world and are silently given probability 1.

Let P denote the initiatedAt/terminatedAt rules along with the *inertia* axiom (3). The SDEs, that is, the ground happensAt and holdsAt predicates serving as input to the system (see Def. 1), are the probabilistic facts. A specific selection of SDEs, i.e., a world, along with P constitute a program L. The probability of L, $P(L)$, is the probability of the world (the product of the probabilities of the included probabilistic facts). If there are n probabilistic facts, there are 2^n different programs L. The ProbLog semantics is defined only for programs that have a unique model, that is, they are locally stratified (Fierens et al. 2015; Przymusinski 1987). Note that local stratification is a standard assumption in EC (Tsilionis, Artikis, and Paliouras 2022) and in CER (Giatrakos et al. 2020) in general.

CE probability computation Recall from Section 2.1 that the CER process depends only on the SDEs falling inside the current window. This means that at each q_i the possible programs L change. To compute the probability that a CE/query q holds at a specific time-point — *success probability* $P(q)$ — we have to find those programs L that their model entails q, i.e. $\mathbf{M}_L \models q$, and sum their probabilities, $P(q) = \sum_{\mathbf{M}_L \models q} P(L)$. The latter, known as weighted model counting (WMC) (Chavira and Darwiche 2008), involves summing through an exponential number of summands, since the number of possible programs L is 2^n.

The computation of $P(q)$ can also be achieved by computing the probability that at least one of the proofs of q is sampled. This can be expressed as:

$$P(q) = P\Big(\bigvee_{p\ \in\ Proofs(q)} \ \bigwedge_{rule\ r\ \in\ p} \ \bigwedge_{b_i\ \in\ body(r)}\Big) \tag{4}$$

Eq. (4) is the probability of a disjunctive normal form (DNF). In EC, this corresponds to computing the probability of the *inertia* axiom (3) for q. However, in order to compute the correct value of $P(q)$, one has to ensure that the different proofs of q are disjoint, meaning that they represent mutually exclusive possible worlds. In the general case, the proofs of q are not disjoint since they may have overlapping conjuncts. Consider, for example, the predicate $\mathsf{happensAt}(walking(P_1),\ T)$ in rule-set (2), which is common among the initiatedAt and terminatedAt rules. Making the proofs disjoint is known as the disjoint-sum problem and is known to be $\sharp P$-hard (Valiant 1979).

Eq. (4) in this form is computationally intractable. ProbLog handles this problem by transforming the DNF into a deterministic decomposable negation normal form (d-DNNF) circuit. This is known as the knowledge compilation step (Darwiche and Marquis 2002). Due to the two proper-

ties of d-DNNF, determinism and decomposability, the probabilistic inference procedure has linear time complexity.

Example 2. *Assume we want to compute the probability of* holdsAt($moving(c_1, c_2)$=true, $t+1$). *Furthermore, consider the following shorthand notation of grounded probabilistic predicates:*

$$
\begin{aligned}
h^{t+1} &\rightarrow \text{holdsAt}(moving(c_1, c_2)\text{=true}, t+1)\\
0.2 :: h^t &\rightarrow \text{holdsAt}(moving(c_1, c_2)\text{=true}, t)\\
1 :: u^{t+1} &\rightarrow next(t, t+1)\\
0.3 :: wlk_1^t &\rightarrow \text{happensAt}(walking(c_1), t)\\
0.6 :: wlk_2^t &\rightarrow \text{happensAt}(walking(c_2), t)\\
1 :: cls_T^t &\rightarrow \text{holdsAt}(close(c_1, c_2)\text{=true}, t)\\
0 :: cls_F^t &\rightarrow \text{holdsAt}(close(c_1, c_2)\text{=false}, t)\\
1 :: so^t &\rightarrow \text{holdsAt}(similarOrientation(c_1, c_2)\text{=true}, t)
\end{aligned}
$$

In the above narrative of probabilistic predicates, all of them, except h^{t+1} (the probability of which we want to compute), are also accompanied by their probability value. The weight of u^{t+1} is 1, since it is a crisp fact, the weight of h^t is assumed to be known and equal to 0.2, and the remaining weights are the probabilities of the corresponding facts. Additionally, we have assumed that $\mathcal{C}$ and $\mathcal{T}$ have two entities and two time-points respectively, and have applied the substitutions $P_1 \mapsto c_1, P_2 \mapsto c_2$, and $T \mapsto t+1$. To compute $P(h^{t+1})$, Prob-EC, through ProbLog, will first construct the d-DNNF circuit presented in Fig. 1. The nodes of the circuit in Fig. 1 correspond to disjunctions and conjunctions. The leaves of the circuit, represented by rectangles in Fig. 1, correspond to the grounded body literals (the above list of predicates) of the inertia axiom (3) *and the* initiatedAt/terminatedAt *rules of rule-set* (2).

The resulting d-DNNF circuit may be even simpler, since ProbLog proceeds to compilation after grounding the program and pruning inactive rules. To compute the probability of the d-DNNF, ProbLog transforms it to an arithmetic circuit by assigning to the leaves the probabilities of the Boolean variables (grounded predicates of Ex. 2) and replacing $\wedge$ and $\vee$ with $\times$ and $+$ operations, respectively (see the parentheses in Fig. 1). By evaluating the arithmetic circuit in a bottom-up fashion until we arrive at the root node, we get the success probability (WMC) of the CE/query q. The value of the root in the arithmetic circuit of Fig. 1, where $q \mapsto$ holdsAt($moving(c_1, c_2)$=true, $t+1$) is:

$$
\begin{aligned}
&P(q) = P(h^{t+1}) =\\
&P(u^{t+1}) \times \Big[\big(P(h^t) \times P(\neg wlk_1^t)\big) + P(wlk_1^t) \times\\
&\Big[\Big[\big[(P(\neg h^t) \times P(wlk_2^t) \times P(cls_T^t) \times P(so^t)) + P(h^t)\big] \times \qquad (5)\\
&P(\neg cls_F^t)\Big] + (P(cls_F^t) \times P(wlk_2^t) \times P(cls_T^t) \times P(so^t))\Big]\Big] =\\
&0.344\,.
\end{aligned}
$$

Notice that the procedure described above has to be performed at each query time q_i, for every time-point

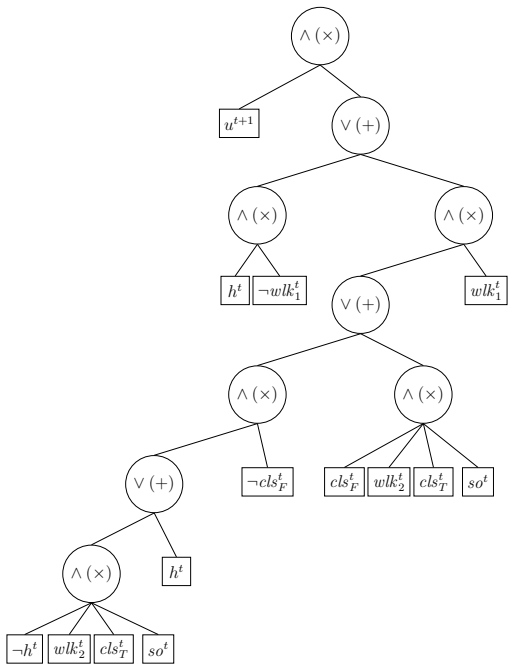

Figure 1: The d-DNNF (arithmetic) circuit of h^{t+1} from Ex. 2.

included in $\mathcal{T}$ (but the first, see Section 2.1). Assume, for example, we want to compute the probability of holdsAt($moving(c_1, c_2)$=true, $t+2$). Then, the constructed circuit at $t+2$ will contain as a sub-circuit the circuit of h^{t+1} in Fig. 1, leading to redundant operations both during compilation and evaluation. Prob-EC includes the compilation mechanism of ProbLog, which in combination with the recursive nature of the *inertia* axiom (3), makes such computations redundant. To overcome this issue, Prob-EC employs a minimal window of size $|\omega|$=2 along with a caching technique, according to which the probability of holdsAt($fl(X, Y)$=v, T_{prev}) is stored in memory, and then it is checked how this probability is affected at the next time-point T by the initiation and termination conditions fired at T_{prev}. This technique simplifies the structure of the arithmetic circuit, constructed at each time-point, by not having to expand it for several past time-points. In Section 5, we show that the performance of Prob-EC is enhanced when $|\omega|$=2 and caching is applied.

3 Probabilistic Linear Algebraic Approach

We present our method, tensor-pEC, for computing the success probabilities (WMC) of the fluents of an EC event description 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-k tensor (k specifies the number of dimensions, where $k > 2$) 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-k tensor $\underline{\mathbf{X}}$, as $\mathbf{x}_i$ and $\underline{\mathbf{X}}_{i_1,\ldots,i_k}$, respectively.

Definition 4 (**mode-(n,m) product**). *Let $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ be two order-k and order-l 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:*

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

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 Prob-EC in Tensor Spaces

The EC language contains the sets of constants, $\mathcal{C}$ and $\mathcal{T}$ (Section 2.1), 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 probabilistic 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 probabilistic predicate r is encoded by an order-3 tensor $\underline{\mathbf{R}} \in [0,1]^{N \times N \times \Omega}$, where:*

$$\underline{\mathbf{R}}_{i,j,k} = \begin{cases} p, & \text{if } P(\bigvee_{Proofs(r)}) = p \text{ for } c_i, c_j, t_k \\ 0, & \nexists \, Proof(r) \text{ for } c_i, c_j, t_k \end{cases}$$
$$\forall\, i,j : 1 \leq i,j \leq N, \forall\, k : 1 \leq k \leq \Omega \ . \quad \blacksquare$$

Element $\underline{\mathbf{R}}_{i,j,k}$ equals p, that is, the probability of the disjunction of proofs of r for variable groundings c_i, c_j, t_k, and 0 if there is no proof of r for these variable groundings. The example below illustrates this encoding.

Example 3. *Assume that $\mathcal{C}$=$\{c_1, c_2\}$ has two entities, say person 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 ground probabilistic EC facts, expressing the probabilities that two persons are close to each other:*

$$0.3 :: \mathsf{holdsAt}(close(c_1, c_2){=}\mathsf{true}, t_1)$$
$$0.6 :: \mathsf{holdsAt}(close(c_1, c_2){=}\mathsf{true}, t_2)$$
$$0.7 :: \mathsf{holdsAt}(close(c_2, c_1){=}\mathsf{true}, t_2) \quad .$$

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

$$\mathbf{c}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ \mathbf{t}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ \underline{\mathbf{C^T}} = \left[\begin{array}{cc|cc} 0 & 0.3 & 0 & 0.6 \\ 0 & 0 & 0.7 & 0 \end{array}\right]. \quad (6)$$

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 $\underline{\mathbf{C^T}}$, we use the notation $\underline{\mathbf{C^T}}_{:,:,i}$. In the tensor representation in (6)*, the value of an element of $\underline{\mathbf{C^T}}$ signifies the probability the predicate to be true for specific groundings of the variables. For example, the probabilistic fact* $0.6 :: \mathsf{holdsAt}(close(c_1, c_2){=}\mathsf{true}, t_2)$ *corresponds to the element $\underline{\mathbf{C^T}}_{1,2,2}$ of $\underline{\mathbf{C^T}}$, and states that the probability of fluent $close =$ true to hold at t_2, for entities c_1 and c_2, is 0.6.*

To query the probability value of a specific variables' grounding, e.g. $\mathsf{holdsAt}(close(c_i, c_j){=}\mathsf{true}, t_k)$, we use the following:

$$\begin{aligned} &\mathsf{holdsAt}(close(c_i, c_j){=}\mathsf{true}, t_k) = \\ &\underline{\mathbf{C^T}} \times_{1,1} \mathbf{c}_i \times_{2,1} \mathbf{c}_j \times_{3,1} \mathbf{t}_k = \\ &\underline{\mathbf{C^T}}_{i,j,k} \in [0,1], \\ &\forall\, i,j : 1 \leq i,j \leq N, \forall\, k : 1 \leq k \leq \Omega \ . \end{aligned} \quad (7)$$

3.3 Probabilistic Inference in Tensor Spaces

To compute the WMC (success probability) of a fluent to hold at each of the time-points specified by $\mathcal{T}$, we need to combine the initiatedAt/terminatedAt rules with the *inertia* axiom (3). In rule (1), we presented the general syntax of initiatedAt and terminatedAt rules, while in Section 2.2 we showed, with the use of an example (see Ex. 2), how Prob-EC transforms the *inertia* axiom (3) into a circuit.

In tensor-pEC, similarly to Prob-EC, we compile the *inertia* axiom (3) to an arithmetic circuit. In contrast to Prob-EC, we compile the *inertia* axiom (3) into a circuit once for each fluent, without employing grounding. This means that the obtained structure includes all the proofs of a fluent as opposed to Prob-EC, where grounding may lead to pruning of proofs (see Section 2.2). Then we evaluate the circuit until the end of the CER process. This compile-once-evaluate-often paradigm has gained a lot of attention recently in probabilistic inference (Maene, Derkinderen, and Martires 2025; Fierens et al. 2015). Before we discuss the evaluation of the compiled circuits, we first need to 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 the probability of a negative literal, we subtract from 1 each element of the tensor encoding the probability of the corresponding positive literal. Consider the negative literal not $\mathsf{happensAt}(a(X,Y), T)$. The tensor $\neg\underline{\mathbf{A}}$ used to represent the probability of 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 4. *The negation of* $\underline{\mathbf{C^T}}$ *from* (6) *is:*

$$\neg\underline{\mathbf{C^T}} = \left[\begin{array}{cc|cc} 1 & 0.7 & 1 & 0.4 \\ 1 & 1 & 0.3 & 1 \end{array}\right]\,.$$

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

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

We denote the probabilities of each literal 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:*

$$\underline{\mathbf{A}} \odot \underline{\mathbf{B}} \in [0,1]^{N\times N\times\Omega}\,. \qquad \blacksquare$$

Example 5. *Consider the following tensors* $\underline{\mathbf{A}}$ *and* $\underline{\mathbf{B}}$*:*

$$\underline{\mathbf{A}} = \left[\begin{array}{cc|cc} 0 & 0.5 & 0 & 0.2 \\ 0.6 & 0 & 0 & 0 \end{array}\right],\ \underline{\mathbf{B}} = \left[\begin{array}{cc|cc} 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.3 & 0 \end{array}\right]\,.$$

Their conjunction would be:

$$\left[\begin{array}{cc|cc} 0 & 0 & 0 & 0.1 \\ 0 & 0 & 0 & 0 \end{array}\right],$$

stating that event a *and fluent* b *can both be true with a probability of 0.1 only at time-point* t_2 *and for entities* c_1 *and* c_2*.*

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:*

$$\underline{\mathbf{A}} + \underline{\mathbf{B}} \in [0,1]^{N\times N\times\Omega}\,. \qquad \blacksquare$$

Notice that in case of tensor disjunction, the value of an element is guaranteed to be inside the interval $[0,1]$, due to the compilation into d-DNNF form (Darwiche and Marquis 2002; Fierens et al. 2015). Next, we discuss the evaluation of the compiled circuits.

The leaves of each circuit are replaced by the tensors encoding the probabilistic predicates. For example, the leaf wlk_1^t in Fig. 1, which corresponds to the atom $\mathsf{happensAt}(walking(c_1),\ t)$, is replaced by a matrix $\mathbf{W} \in [0,1]^{N\times\Omega}$, that now encodes the probabilities of all the possible variable groundings of predicate $\mathsf{happensAt}(walking(P_1),\ T)$. The possible groundings also include leaf wlk_2^t, referring to atom $\mathsf{happensAt}(walking(c_2),\ t)$. If a leaf corresponds to a negative predicate, e.g., $\neg wlk_1^t$, the negative version of the matrix/tensor is used, i.e., $\neg\mathbf{W} \in [0,1]^{N\times\Omega}$, computed as per Def. 6.

Conjunction (multiplication) and disjunction (addition) nodes of the circuit are handled by the Hadamard product and tensor addition, respectively. However, Hadamard product and addition are defined only on tensors of the same order and size. Matrix $\mathbf{W}$, discussed above, is an order lower than the tensors representing the remaining leaves of the circuit in Fig. 1. To deal with this issue, we perform the following outer product:

$$\mathbf{W} \circ \mathbf{1}^N = \underline{\mathbf{W^1}} \in [0,1]^{N\times N\times\Omega}\,,$$

which transforms $\mathbf{W}$ to an order-3 tensor, i.e., $\underline{\mathbf{W^1}}$. $\underline{\mathbf{W^1}}$ is now encoding the probabilities of the artificial predicate $\mathsf{happensAt}(walking(P_1,P_2),\ T)$. Regardless the grounding of P_2, the elements of $\underline{\mathbf{W^1}}$ correspond to the groundings of predicate $\mathsf{happensAt}(walking(P_1),\ T)$. We use the superscript 1 in $\underline{\mathbf{W^1}}$, to differentiate from tensor $\underline{\mathbf{W^2}}$. $\underline{\mathbf{W^2}}$ is the result of pre-multiplying matrix $\mathbf{W}$ with the all-ones vector $\mathbf{1}^N$. $\underline{\mathbf{W^2}}$, similarly to $\underline{\mathbf{W^1}}$, represents the same artificial predicate, but its elements correspond now to groundings of predicate $\mathsf{happensAt}(walking(P_2),\ T)$. The reason we need both tensors $\underline{\mathbf{W^1}}$ and $\underline{\mathbf{W^2}}$, is the fact that the probabilities of the same predicate and at the same time-point, but for different entity substitutions, have to be multiplied or added in the arithmetic circuit of Fig. 1, e.g., $P(wlk_1^t) \times P(wlk_2^t)$.

However, simply replacing the leaves by the corresponding tensors/matrices and applying the operations dictated by the circuit (see Eq. (5)), does not produce the WMC of a fluent to hold. In the next section, we elaborate on how to obtain WMC algebraically.

4 Algebraic Weighted Model Counting

In Eq. (5), we have shown the series of operations needed to evaluate the circuit of Fig. 1. This evaluation computes the probability of the fluent to hold at the next time-point, i.e., $P(h^{t+1})$, given the probability of the fluent to hold at the previous time-point, i.e., $P(h^t)$. In general, when compiling the *inertia* axiom (3) for a fluent to an arithmetic circuit, one can observe by inspecting the structure, that $P(h^t)$ can be taken out as a common factor for some of the terms. Let $P(d^t)$ denote the multiplicand of $P(h^t)$ and $P(s^t)$ the rest of the terms. Then, Eq. (5), but also any evaluation of a circuit referring to a fluent, can be written as:

$$P(h^{t+1}) = P(s^t) \times P(u^{t+1}) + P(h^t) \times P(d^t) \times P(u^{t+1})\,. \tag{8}$$

Notice that $P(s^t)$ and $P(d^t)$ are the result of multiplications and additions. For example, $P(s^t)$ and $P(d^t)$ of Eq. (5), after simplifying terms, are the following:

$$\begin{aligned} P(s^t) &= P(wlk_1^t) \times P(wlk_2^t) \times P(cls_T^t) \times P(so^t)\,, \\ P(d^t) &= 1 + (P(s^t) \times P(cls_F^t)) - P(s^t) - P(wlk_1^t) \times P(cls_F^t)\,. \end{aligned} \tag{9}$$

In the algebraic approach, $P(s^t)$ and $P(d^t)$ are represented by tensors $\underline{\mathbf{S}}, \underline{\mathbf{D}} \in [0,1]^{N\times N\times\Omega}$, respectively. Let, also,

$\underline{\mathbf{W^1}}, \underline{\mathbf{W^2}}, \underline{\mathbf{C^T}}, \underline{\mathbf{C^F}}, \underline{\mathbf{O}} \in [0,1]^{N\times N\times\Omega}$ be the tensors encoding $P(wlk_1^t), P(wlk_2^t), P(cls_T^t), P(cls_F^t)$ and $P(so^t)$, respectively. $\underline{\mathbf{S}}$ and $\underline{\mathbf{D}}$ are produced by employing the algebraic operations for the logical connectives and negation, discussed in Section 3.3, and thus, Eq. (9), would be computed in tensor spaces as:

$$\begin{aligned}\underline{\mathbf{S}} &= \underline{\mathbf{W^1}} \odot \underline{\mathbf{W^2}} \odot \underline{\mathbf{C^T}} \odot \underline{\mathbf{O}}\,,\\ \underline{\mathbf{D}} &= \underline{\mathbf{1}} + (\underline{\mathbf{S}} \odot \underline{\mathbf{C^F}}) - \underline{\mathbf{S}} - (\underline{\mathbf{W^1}} \odot \underline{\mathbf{C^F}})\,,\end{aligned} \tag{10}$$

where $\underline{\mathbf{1}} \in \{1\}^{N\times N\times\Omega}$ is an all-ones order-3 tensor.

$P(u^{t+1})$ in Eq. (8) denotes the probability of predicate $next(T_{prev}, T)$ in *inertia* axiom (3), which is always 1, except for the first time-point, i.e., $T \mapsto t_1$. Recall from Section 2.1, that $next$ signifies that the next time-point of T_{prev} is T. In Prob-EC, $next$ serves only the grounding of the time variable and it does not affect the result of the evaluation of the circuit (see Section 2.2). In the algebraic approach, since both variables in $next$ take values from $\mathcal{T}$, it is encoded 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. Note that the multiplications with $P(u^{t+1})$ in Eq. (8), e.g., $P(s^t) \times P(u^{t+1})$, cannot be performed with the Hadamard product in tensor-pEC, i.e., $\underline{\mathbf{S}} \odot \mathbf{U}$, since the participating tensors/matrices have to be of the same order and size, and encode predicates sharing the same variables at all positions. To resolve this, we use the mode-(n,m) product of a tensor with $\mathbf{U}$. Multiplying a tensor $\underline{\mathbf{A}} \in [0,1]^{N\times N\times\Omega}$ with matrix $\mathbf{U}$, i.e., $\underline{\mathbf{A}} \times_{3,1} \mathbf{U}$, results in the shifting of elements of $\underline{\mathbf{A}}$ along the temporal dimension, that is, the first temporal slice is a matrix of zeros, $\underline{\mathbf{A}_{:,:,1}} = \mathbf{0}^N \circ \mathbf{0}^N$, where $\mathbf{0}^N$ an all-zeros vector.

Finally, let $\underline{\mathbf{H}}$ be the tensor encoding the success probabilities (WMC) of a fluent *fl* to hold. $P(h^{t+1})$ and $P(h^t)$ of Eq. (8) are elements of $\underline{\mathbf{H}}$. Everything is now in place and we can proceed with the correct substitutions of probabilities and arithmetic operations in Eq. (8), with tensors/matrices and algebraic operations, respectively. Eq. (8) can be expressed algebraically as:

$$\begin{aligned}&\underline{\mathbf{H}} = \underline{\mathbf{S}} \times_{3,1} \mathbf{U} + \big(\underline{\mathbf{H}} \odot \underline{\mathbf{D}}\big) \times_{3,1} \mathbf{U} \Leftrightarrow\\ &\underline{\mathbf{H}} - \big(\underline{\mathbf{H}} \odot \underline{\mathbf{D}}\big) \times_{3,1} \mathbf{U} = \underline{\mathbf{S}} \times_{3,1} \mathbf{U}\,.\end{aligned} \tag{11}$$

Eq. (11) is a first-order difference (recursive) equation and our goal is to compute the unknown tensor $\underline{\mathbf{H}}$. In this form though, Eq. (11) cannot be solved. Unfolding Eq. (11) for every element of tensors $\underline{\mathbf{H}}, \underline{\mathbf{S}}, \underline{\mathbf{D}}$, 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:

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

Notice that the initial condition $\underline{\mathbf{H}}_{i,j,1} = 0,\ 1 \le i,j \le N$, in agreement with *inertia* axiom (3), states that at the first time-point t_1, regardless the pair of entities c_i, c_j, no fluent can hold, and thus, the probability is 0. The above system can be written in matrix form as:

$$\underbrace{\begin{bmatrix} 1 & & & \\ -\underline{\mathbf{D}}_{1,1,1} & 1 & & \\ & \ddots & \ddots & \\ & & -\underline{\mathbf{D}}_{N,N,\Omega-1} & 1 \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} \underline{\mathbf{H}}_{1,1,1} \\ \underline{\mathbf{H}}_{1,1,2} \\ \vdots \\ \underline{\mathbf{H}}_{N,N,\Omega} \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} 0 \\ \underline{\mathbf{S}}_{1,1,1} \\ \vdots \\ \underline{\mathbf{S}}_{N,N,\Omega-1} \end{bmatrix}}_{\mathbf{b}}, \tag{12}$$

where $\mathbf{G} \in \mathbb{R}^{N^2\Omega\times N^2\Omega}$ is the coefficients matrix, and $\mathbf{h},\ \mathbf{b} \in [0,1]^{N^2\Omega}$, are column vectors. Our goal is to solve Eq. (12) for $\mathbf{h}$, i.e., the probabilities of a fluent to hold at each time-point and for each pair of entities.

To construct the matrix equation (12), 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, $vec[\underline{\mathbf{A}}] : \underline{\mathbf{A}} \in \mathbb{R}^{N\times N\times\Omega} \rightarrow \mathbf{a} \in \mathbb{R}^{N^2\Omega}$. Then, the operations to produce $\mathbf{G}$ and $\mathbf{b}$ in Eq. (12), are the following:

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

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

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 $\underline{\mathbf{D}}$ multiplied by -1, and all the remaining elements are set to 0. Notice that, due to multiplication of $vec\big[\underline{\mathbf{D}}\big]$ by -1, $\mathbf{G} \notin [0,1]^{N^2\Omega\times N^2\Omega}$ but $\mathbf{G} \in [-1,1]^{N^2\Omega\times N^2\Omega}$. Vector $\mathbf{b}$, in (b), is the vectorization of tensor $\underline{\mathbf{S}}$, 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. (12) has a unique solution, that can be expressed formally for vector $\mathbf{h}$ and tensor $\underline{\mathbf{H}}$, both encoding the instantaneous probabilities of $fl(X,Y)=v$ to hold, as per Def. 9.

Definition 9 (Algebraic WMC)**.** *The success probabilities (WMC) of a fluent-value pair, are computed by:*

$$\begin{aligned}&\mathbf{h} = \mathbf{G}^{-1}\mathbf{b},\\ &\underline{\mathbf{H}} = vec^{-1}\big[\mathbf{h}\big] \in [0,1]^{N\times N\times\Omega}\,.\end{aligned} \tag{13}$$

■

In Def. 9, $vec^{-1}[\cdot]$ 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. Def. 9 may be extended for tensors of any order, i.e., fluents with arity > 2.

The process described so far is repeated for every fluent-value pair by following the ordering imposed by stratification. This is also the case for Prob-EC (see Section 2.2). The tensors of each stratum are cached and propagated to higher strata. At the end, each tensor is the WMC of the fluent it encodes.

Proposition 1 (Correctness)**.** *The unique solution of Eq.* (12)*, computed by Eq.* (13)*, coincides with the WMC of a fluent-value pair to hold, as computed by Prob-EC.* ♦

The proof may be found in the Supplementary Material[1].

Proposition 2 (Complexity)**.** *The time complexity of solving Eq.* (12) *is* $\mathcal{O}(N^{k-1}\Omega)$ *for order-*k *tensors* (Demmel 1997). ♦

Eq. (12) requires the construction of the coefficients matrix $\mathbf{G}$ and vector $\mathbf{b}$. The first sub-diagonal of $\mathbf{G}$ depends on tensor $\underline{\mathbf{D}}$, while $\mathbf{b}$ depends on tensor $\underline{\mathbf{S}}$ shifted (mode-(3,1) product) by matrix $\mathbf{U}$. $\underline{\mathbf{S}}$ and $\underline{\mathbf{D}}$ are produced by evaluating the arithmetic circuit of the fluent and require linear time. The time complexity of the mode-(3,1) product $\underline{\mathbf{S}} \times_{3,1} \mathbf{U}$, is $\mathcal{O}(N^{p-1}\Omega^2)$. In Prob-EC, similarly, the WMC depends on the size of the arithmetic circuit, and it has linear time complexity. However, Prob-EC has to perform the knowledge compilation step ($\sharp P$-hard) for each fluent at each application of the window, as opposed to the tensor method that employs the compile-once-evaluate-often paradigm. Hence, tensor-pEC is theoretically bound by lower complexity.

Additionally, the performance of tensor-pEC 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 usual case is for the probabilistic facts (SDEs) to be only a small fraction of all the possible predicate groundings. In Prob-EC, only the SDEs appearing in the input are considered. On the other hand, in tensor-pEC, due to the employed representation, all the possible predicate groundings are encoded in tensors, resulting in many zero elements (see Def. 5). By exploiting the sparse structure of tensors we can avoid unnecessary calculations and improve performance by not examining null elements (Nguyen, Inoue, and Sakama 2022).

5 Empirical Analysis

5.1 Experimental Setup

Tensor-pEC is implemented in Python. We compare the performance of tensor-pEC against that of Prob-EC. Prob-EC is implemented in ProbLog (Fierens et al. 2015), which is mostly written in Python. The source code of both methods and the datasets, are available in the Supplementary Material[1]. The experiments were performed on a single core, i.e., no parallelization was used, on a desktop computer with Intel® Core™ i7-4770×8 and 16 GB of RAM, running Ubuntu 24.04.2 LTS and Python 3.12.2.

We selected Prob-EC as a comparison method, since it has been shown to be more efficient for CER (Mantenoglou, Artikis, and Paliouras 2023) than other probabilistic EC point-based benchmark systems, including Probabilistic EC (PEC) (D'Asaro et al. 2020) and Simplified EC (SEC) (McAreavey et al. 2017). Another EC point-based system is MLN-EC (Skarlatidis et al. 2015b), where the CEs are expressed through Markov Logic Networks (MLNs). This framework is unsuitable for CER as well, since grounding results in exponential networks in size. Moreover, MLN-EC performs approximate reasoning as opposed to the exact inference of Prob-EC and tensor-pEC.

The probabilistic CER process involves the computation and caching of the instantaneous probabilities of fluent-value pairs, expressing CEs, to hold. We used CAVIAR[2], a benchmark human activity recognition dataset. This dataset includes annotated frames of videos, where people walk around, meet one another, fight, and so on — recall 'moving' from rule-set (2). The input data correspond to SDEs, e.g., $\mathsf{happensAt}(walking(P_1),\ T)$ in rule-set (2) along with contextual information such as $\mathsf{holdsAt}(close(P_1, P_2)\text{=}\mathsf{true},\ T)$. To make the dataset suitable for probabilistic CER, we followed (Skarlatidis et al. 2015a) and injected noise through a Gamma distribution to the input, producing two versions of the dataset. The first one, called 'caviar smooth', assigns probability values only to SDEs. The second, 'caviar strong', adds also noise, apart from SDEs, to contextual data. Moreover, spurious SDEs, not belonging to the original dataset, have been added using a uniform distribution.

We also employed a dataset from the maritime domain. In maritime monitoring, CER concerns the recognition of composite maritime events 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, Chatzikokolakis, and Zissis 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 SDEs to our system and noise has been injected to them by following the approach of (Mantenoglou, Artikis, and Paliouras 2023). We employed a publicly available dataset, concerning vessels sailing in the Atlantic Ocean around the port of Brest, France.

To simulate a streaming behavior, the datasets are stored in CSV files and processed periodically in chunks according to the window ω specification. Notice that, given a constant window ω, the number of SDEs varies from window to window and consequently, the number of entities (the N constants of set $\mathcal{C}$) changes, while the Ω time-points of set $\mathcal{T}$ (size of ω) remain unchanged. Consecutive windows overlap by one time-point. To boost the performance of tensor-pEC, we use sparse representations for the tensors encoding fluents. Since efficient implementations of operations, such as the Hadamard or the mode-(n, m) product, on sparse tensors do not exist, 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 $\mathbf{F} \in [0, 1]^{N^2\times\Omega}$. Finally, to favor the performance of both methods, we use a CE probability threshold $\theta = 10^{-6}$ and make every value lower than θ equal to 0. This means that Prob-EC does not have to cache the thresholded prob-

[1]https://github.com/eftsilio/tensor-pEC

[2]https://groups.inf.ed.ac.uk/vision/DATASETS/CAVIAR/CAVIARDATA1/

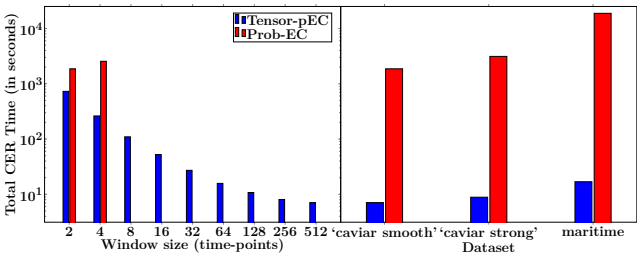

Figure 2: Tensor-pEC vs. Prob-EC. Left: 'caviar smooth'. Right: All datasets.

abilities, while tensor-pEC increases the sparsity of the tensors.

5.2 Experimental Results

We conducted two experiments to test the performance of tensor-pEC and Prob-EC. In the first experiment we use the 'caviar smooth' dataset and vary the window size from 2 to 512 time-points in order to find the best window configuration for each method. Fig. 2 (left) displays the results, where the x-axis states the window size and the y-axis corresponds to total recognition time (in log scale). Tensor-pEC achieves a performance gain in all windows, and this gain becomes more profound as the window size increases. The reasoning time for tensor-pEC decreases with increasing window sizes, due to the fewer times it has to construct the tensors. Prob-EC exhibits the opposite pattern, i.e., as the window size increases the performance deteriorates; for windows of size $|\omega|{>}4$ the available memory is consumed and the CER process is not completed. As discussed in Section 2.2, and experimentally confirmed in Fig. 2 (left), the use of a window $|\omega|{=}2$ along with caching, favors the performance of Prob-EC, since the knowledge compilation step pertains to a single time-point and the created circuits are smaller in size.

The second experiment employs the best window configuration, observed in the first experiment, for each method, i.e., $|\omega|{=}2$ for Prob-EC and $|\omega|{=}512$ for tensor-pEC, and evaluates them on all datasets. Fig. 2 (right) shows the total CER time for the CAVIAR and Brest datasets (x-axis). Tensor-pEC is the preferred option in all datasets, highlighting its efficiency in comparison to Prob-EC. Regarding Prob-EC, the knowledge compilation step requires $\approx 98\%$ of the total time in all datasets. Even without this step, tensor-pEC continues to be the best method. Note that applying the compile-once-evaluate-often paradigm to Prob-EC, and consequently to ProbLog, is not straightforward, since, apart from evaluating the circuits for different probabilities in the leaves, we also have to take care of the unification, which, in the current setup, takes place during the grounding of the proofs and before the compilation stage.

The memory consumption is negligible for both methods in all datasets in these window settings.

6 Related Work

Several EC implementations for logical reasoning over traces of events have been proposed in the literature, including symbolic ones (Artikis, Sergot, and Paliouras 2015; Tsilionis, Artikis, and Paliouras 2022) and algebraic ones (Tsilionis, Artikis, and Paliouras 2024). These approaches may be scalable, but do not consider any type of uncertainty, and thus, the recognition is characterized by limited confidence. Prob-EC (Skarlatidis et al. 2015a) is a benchmark symbolic probabilistic EC point-based CER system based on ProbLog (Fierens et al. 2015) (see Section 2.2). In Section 5.2, we have included Prob-EC as a comparison method due to its efficiency (Mantenoglou, Artikis, and Paliouras 2023), and have used high-volume/velocity temporal datasets as well as large event descriptions. In these extreme settings, tensor-pEC improves the inference time by orders of magnitude. Prob-EC performs, at each application of the window, the costly operation of knowledge compilation before computing the CE probabilities. In tensor-pEC, compilation is applied once (Maene, Derkinderen, and Martires 2025), and the produced circuit is expressed as a linear recursive equation. Then, solving the equation (Def. 9) produces significantly faster, compared to Prob-EC, the WMC of CEs, by exploiting the sparse structure of tensors and efficient implementations of algebraic operations.

A work closely related to ours, in the sense that it performs probabilistic inference in vector spaces, is TensorLog (TL) (Cohen, Yang, and Mazaitis 2020). TL's main goal is to combine inference and learning by means of deep learning infrastructures. To this end, TL does not support the *possible world* semantics, and thus, it is more appropriate for approximate inference as opposed to the exact inference of tensor-pEC. Furthermore, TL's language cannot express the EC dialect employed by tensor-pEC. TL utilizes a single set of constants and it represents unary and binary predicates by vectors and square matrices, respectively. In contrast, our method, by incorporating time in the representations, it uses an additional set of constants and is not restricted to square/cubical matrices/tensors. Moreover, our approach can be extended to tensors (predicates) of any order (arity).

7 Summary and Future Work

We proposed a linear algebraic approach for probabilistic EC reasoning. We represent EC probabilistic predicates as tensors and demonstrate that the success probabilities (WMC) of fluents (CEs) of any arity can be assessed by solving a linear recursive equation. The scalability of our system is empirically demonstrated on real-world streaming data from human activity recognition and the maritime domain. Additionally, our approach improves the performance of the symbolical probabilistic EC, by orders of magnitude. An interesting future work direction would be to develop a neuro-symbolic (NeSy) EC system for CER. Most NeSy systems for CER apply symbolic solvers on the logical layer (Roig Vilamala et al. 2023), such as, DeepProbLog (Manhaeve et al. 2021), which can be computationally highly expensive and inefficient. By embedding the logical layer into tensor spaces and leveraging our linear algebraic approach for inference, we can enhance the overall performance of a NeSy CER system. 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 CREXDATA "Critical Action Planning over Extreme-Scale Data" project (No 101092749), which has received funding from the EU Horizon Europe research and innovation programme.

References

Alevizos, E.; Skarlatidis, A.; Artikis, A.; and Paliouras, G. 2017. Probabilistic complex event recognition: A survey. *ACM Comput. Surv.* 50(5):71:1–71:31.

Artikis, A.; Makris, E.; and Paliouras, G. 2021. A probabilistic interval-based event calculus for activity recognition. *Ann. Math. Artif. Intell.* 89(1-2):29–52.

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

Bereta, K.; Chatzikokolakis, K.; and Zissis, D. 2021. Maritime reporting systems. In Artikis, A., and Zissis, D., eds., *Guide to Maritime Informatics*. Springer. 3–30.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172(6):772–799.

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

Cohen, W.; Yang, F.; and Mazaitis, K. 2020. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *Journal of Artificial Intelligence Research* 67:285–325.

Darwiche, A., and Marquis, P. 2002. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research* 17:229–264.

D'Asaro, F. A.; Bikakis, A.; Dickens, L.; and Miller, R. 2020. Probabilistic reasoning about epistemic action narratives. *Artificial Intelligence* 287:103352.

Demmel, J. W. 1997. *Applied Numerical Linear Algebra*. USA: Society for Industrial and Applied Mathematics.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming* 15:358–401.

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.

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

Maene, J.; Derkinderen, V.; and Martires, P. Z. D. 2025. KLay: Accelerating arithmetic circuits for neurosymbolic AI. In *The Thirteenth International Conference on Learning Representations*.

Manhaeve, R.; Dumančić, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2021. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence* 298:103504.

Mantenoglou, P.; Artikis, A.; and Paliouras, G. 2023. Online event recognition over noisy data streams. *Int. J. Approx. Reason.* 161:108993.

McAreavey, K.; Bauters, K.; Liu, W.; and Hong, J. 2017. The event calculus in probabilistic logic programming with annotated disjunctions. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, 105–113. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Nguyen, T. Q.; Inoue, K.; and Sakama, C. 2022. Enhancing linear algebraic computation of logic programs using sparse representation. *New Gen. Comput.* 40(1):225–254.

Patroumpas, K.; Alevizos, E.; Artikis, A.; Vodas, M.; Pelekis, N.; and Theodoridis, Y. 2017. Online event recognition from moving vessel trajectories. *GeoInformatica* 21(2):389–427.

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

Roig Vilamala, M.; Xing, T.; Taylor, H.; Garcia, L.; Srivastava, M.; Kaplan, L.; Preece, A.; Kimmig, A.; and Cerutti, F. 2023. Deepprobcep: A neuro-symbolic approach for complex event processing in adversarial settings. *Expert Systems with Applications* 215:119376.

Sakama, C.; Inoue, K.; and Sato, T. 2021. Logic programming in tensor spaces. *Annals of Mathematics and Artificial Intelligence* 89.

Sato, T. 2017a. 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.

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

Skarlatidis, A.; Artikis, A.; Filipou, J.; and Paliouras, G. 2015a. A probabilistic logic programming event calculus. *Theory Pract. Log. Program.* 15(2):213–245.

Skarlatidis, A.; Paliouras, G.; Artikis, A.; and Vouros, G. A. 2015b. Probabilistic event calculus for event recognition. *ACM Trans. Comput. Logic* 16(2).

Tsilionis, E.; Artikis, A.; and Paliouras, G. 2022. Incremental event calculus for run-time reasoning. *J. Artif. Intell. Res.* 73:967–1023.

Tsilionis, E.; Artikis, A.; and Paliouras, G. 2024. A tensor-based formalization of the event calculus. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, 3584–3592. ijcai.org.

Valiant, L. 1979. The complexity of computing the permanent. *Theoretical Computer Science* 8(2):189–201.