Event Queries on Correlated Probabilistic Streams

Christopher Ré, Julie Letchner, Magdalena Balazinska and Dan Suciu
Dept. of Computer Science and Engineering,University of Washington
Seattle, WA, USA
{chrisre,letchner,magda,suciu}@cs.washington.edu

ABSTRACT

A major problem in detecting events in streams of data is that the data can be imprecise (*e.g.* RFID data). However, current state-of-the-art event detection systems such as Cayuga [14], SASE [46] or SnoopIB[1], assume the data is *precise*. Noise in the data can be captured using techniques such as hidden Markov models. Inference on these models creates streams of probabilistic events which cannot be directly queried by existing systems. To address this challenge we propose Lahar[1], an event processing system for probabilistic event streams. By exploiting the probabilistic nature of the data, Lahar yields a much higher recall and precision than deterministic techniques operating over only the most probable tuples. By using a novel static analysis and novel algorithms, Lahar processes data orders of magnitude more efficiently than a naïve approach based on sampling. In this paper, we present Lahar's static analysis and core algorithms. We demonstrate the quality and performance of our approach through experiments with our prototype implementation and comparisons with alternate methods.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Systems—*Query processing*; G.3 [**Mathematics of Computing**]: Probability and Statistics; F.4.1 [**Mathematical Logic**]

Keywords

Query Processing, Hidden Markov Models, Probabilistic Databases, Streams

General Terms

Algorithms, Management, Performance, Theory

[1]A lahar is a stream of water mixed with dirt that flows down the side of a volcano, typically after an eruption such as that of Mount St. Helens in 1980. The Lahar system manages dirty streams of probabilistic data.

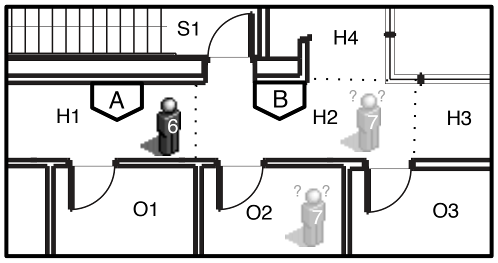


Figure 1: At time 6 **Joe is in hallway** $H1$ **and is read by antenna** A**, so we are sure of his precise location. At time** 7**, we do not receive any reading and so our knowledge of his location is *imprecise*: has he has entered his office,** $O2$**, or continued down the hall to** $H2$ **and simply been missed by antenna** B**?**

1. INTRODUCTION

In this paper we study *event queries* on probabilistic data streams. The data is streaming continuously from real-time data sources such as sensors [39], ticker feeds [37], or network monitors [5], or from historical data sources such as a sensor data archive [11], a ticker history [37], or a network flow database [5]. Applications need to extract complex events (often user-specified) from these streams of low-level atomic events. Such applications include supply chain management [43], financial services [14], business activity monitoring [21], elder care [27, 32], and various pervasive computing applications [12, 25, 28]

Rich query languages supported by existing event detection engines such as Cayuga [14], SASE [46] or SnoopIB [1] are designed to extract sophisticated patterns from event streams. They support powerful constructs such as joins, regular expressions, selections, and projections. However, these languages require the data to be precise. In many of the above application domains, the data is instead *imprecise* due to either errors (*e.g.* missed readings in an RFID deployment [20, 23]), or because the data is the output of inference or predictive algorithms [12, 27, 32]. To address this problem, we propose Lahar that supports complex event queries on imprecise data streams.

1.1 Motivation

Our primary motivating application is an RFID deployment as can be found inside a supply chain or department store [6], an office building [40], or a hospital [38]. In an RFID application, a fundamental relation is the location of people and objects over time. Ideally, we would query this relation as a stream of tuples with schema At(person, location, time). A tuple in the At relation, (‘Joe’, ‘Room 326’, 10:05am) is called an *event* and tells us that Joe was in room 326 at 10:05am. RFID-based applications are typically interested in transforming sequences of such low-level events into higher-level events: *e.g.* “Joe is getting coffee”, “A database group meeting just started”. These can be expressed as *complex event queries* in a high-level, declarative query language such as Cayuga [14], SASE [46], or SnoopIB [1]

RFID deployments, however, produce data that is imprecise for three main reasons: *conflicting readings*, *e.g.* Alice is read by two adjacent antennas, what is her true location? [20]; *missed readings*, *e.g.* readers commonly detect only about 60%-70% of tags in their vicinity [4, 20]; and *granularity mismatch*, an application queries about offices, but the system only provides information about sensors. Fig. 1 illustrates the problem. In this scenario, a user, Joe, wearing an RFID tag, is walking through a building instrumented with RFID readers. At time 6, antenna A has detected Joe’s tag, removing almost all uncertainty about Joe’s location. However, there is no reading for Joe at time 7, and so there are at least two possible options. The first is that Joe could have gone into his office, $O2$, where there are no sensors. Alternatively, he could have continued down the hallway, and his tag was missed by antenna B. In large-scale RFID deployments, read rates vary widely between 10% and 90% [44], making it impossible to distinguish between the various possible cases.

A common approach to dealing with such imprecise data is to build a model of the data and use raw readings as input to the model. For our application, the standard approach is to use a temporal graphical model, the simplest of which is the hidden Markov model or *HMM*. In general, HMMs are used to infer a hidden state based on a sequence of observations, *e.g.* the location of a person based on sensor readings. In a real-time application we can use an HMM to produce a distribution over Joe’s location at each timestep. In an archived application we can continue to use HMMs, but can now leverage more sophisticated techniques such as smoothing [26]. Smoothing not only provides a more accurate distribution over Joe’s location at each timestep, but also provides correlations between Joe’s locations at different timesteps. For example, if we determine that Joe really went into his office at $t = 7$, it is more likely that he will continue to stay in his office at $t = 8$. In other words, Joe’s location at time t is *correlated* with Joe’s location at time $t + 1$.

Answering simple queries in a declarative language over the probabilistic, correlated output of an HMM has previously been studied [22], but only for queries that do not relate different points in time. Many applications require sequences, projections and joins, and so this limited query model is insufficient to meet their needs. In this paper we propose instead to process complex event queries on correlated, probabilistic streams. Our query language is a subset of Cayuga and supports joins, selections, projections, and regular expressions. The query semantics requires complex probabilistic computations at runtime, and our main contribution consists of developing techniques to perform these computations efficiently on archived and streaming data.

More precisely, we identify four increasingly general classes of queries and develop new algorithms for each of them. *Regular Queries* consist of selections and regular expressions, and can be evaluated in streaming fashion; *Extended Regular Queries* add projections and joins across time and can also be evaluated in a streaming manner; *Safe Queries* add more complicated interleavings of projections and sequencing, which require a more sophisticated *probabilistic stream algebra*, and can be efficiently executed on bounded, archived data; and finally, *Unsafe Queries* are provably hard to evaluate and for these we develop an approximation algorithm.

An alternate approach is to discard the probabilities and correlations produced by the low-level inference technique and simply choose the single most likely location at each timestep. Cayuga queries can then be run directly on the now-deterministic stream using their standard semantics. However, we experimentally verify that our approach yields both a higher recall and higher precision than this approach for complex event queries, *e.g.* over 30% improvement in both precision and recall.

Another approach is to encode all queries as higher-level goals directly in the HMM [12, 27, 32]. The limitation of this approach is that the set of possible queries is predefined by the hard-coded model. A third alternate approach is to store the streams in a probabilistic relational database [3, 9]; however, probabilistic databases do not operate on sequences or streams.

Finally, our techniques are more generally applicable than RFID tracking. For example, elder care (or care of the cognitively impaired) applications require that the system *infer* the elder’s activities over time from a variety of sensor readings [27, 32, 41]. Caregivers may want to query these activity streams [41]: did the elder take her medicine today? Did she brush her teeth before going to bed? A variety of other applications use HMMs and could similarly benefit from our system (*e.g.*, [12]).

1.2 Contributions

Our main contributions are a set of algorithms for efficiently processing queries on correlated probabilistic streams. Our approach allows for significantly increased precision and an increase. It is efficient in practice, processing safe queries at a speed of over 100 Ktuples/sec. In Sec. 3, we give our core algorithmic contributions, all of which are implemented in Lahar. We briefly summarize them below:

1. In Sec. 3.1, we provide an efficient algorithm for processing Regular Queries over probabilistic event streams. These streams can contain simultaneous events and correlations across time.
2. In Sec. 3.2, we provide an efficient algorithm for Extended Regular Queries. We also provide a sound and complete static analysis to identify extended regular queries.
3. In Sec. 3.3, we provide an algorithm based on a novel algebra for probabilistic streams that allows us to identify and efficiently process the more general class of Safe Queries.
4. In Sec. 3.4, we give evidence that Safe Queries are the broadest class of queries that can be computed efficiently: We show that even minor extensions lead to intractability, *i.e.* $\sharp\mathcal{P}$-hardness.
5. In Sec. 3.5, we present a general algorithm based on naïve random sampling. It can process any query, but is orders of magnitude slower than our other algorithms.

In Sec. 4, we experimentally verify the quality and performance of our system on a variety of simulated and real RFID traces and scenarios.

2. PRELIMINARIES

In this section we present our data model, query language, and probability model.

2.1 Data Model

An **event** is a tuple e conforming to the schema EventType(ID, a_1, …, a_n, T), with two distinguished attributes: an event key (ID) and a timestamp (T). e represents some real world event such as “Joe got coffee in room 326 at 10:05am”. To indicate the key we underline it, *e.g.* person is the key in the event

GotCoffee($\underline{person}, room, \mathtt{T}$). The attribute T denotes the time when the event ended (*e.g.*, the time when Joe got back from the coffee room with his coffee). A **stream** (or sequence) of events is a set of events of the same type with the same event key (ID), but distinct timestamps (T). An **event database** consists of several streams of events and, optionally, standard relations. Events are continuously appended to the streams. We assume that the value of the attribute T also corresponds to the time when the event tuple is inserted into the database.

A snapshot of the database taken at time n thus contains a set of stream prefixes. We call each such snapshot a **world** W and denote it as a sequence of sets of tuples: $W = (w_1, \ldots, w_n)$ where $w_i \overset{\text{def}}{=} \{e \mid e \in W, e[T] = i\}$ (each w_i is the set of events inserted into the database at time i).

2.2 Event Query Language

Our query language is a strict subset of Cayuga [14], consisting of selections, sequencing, joins, and Kleene plus. We describe it here briefly. To define event queries, we first define a **subgoal**, a **condition**, and a **base query**.

A **subgoal** is a relational symbol with a list of variables and/or constants *without* T. For example, given a relation At($\underline{person}, location, \mathtt{T}$), then At($\underline{x}$, 'Room201') is a subgoal that denotes an event (*i.e.*, a tuple from At) where a person visited room 201 at some time. A **condition** θ is a complex Boolean expression over variables, *e.g.* $y > 20$, or Hall(z). The latter is a relational query checking whether z occurs in the relation Hall.

D 2.1. *A* ***base query****, bq, is either $\sigma_\theta(g)$ (a subgoal with a predicate) or $(\sigma_\theta(g))^+\langle V, \theta_2\rangle$ (parametrized Kleene plus). Here, V denotes the set of variables that are shared across the unfoldings of the Kleene plus and θ_2 is a predicate applied to each unfolding. An* ***event query****, or simply query, is defined recursively as a base query, or if q_1 is a query and bq is a base query, then $q = q_1; bq$ (sequence) and $\sigma_\theta(q_1)$ (selection) are queries as well.*

We define a query in two stages because, in our language, we restrict all sequencing to be left-associative: $E_1; E_2; E_3$ means $(E_1; E_2); E_3$. We don't allow $E_1; (E_2; E_3)$ as in Cayuga [45]. We write **goal**(q) for the set of subgoals in q. We write **var**(θ), **var**(g), **var**(q) for the set of variables in a condition, a subgoal, or a query.

Example 2.2 The simple query from the introduction: "Joe got coffee" can be expressed as a sequence of three events: (1) Joe is in his office, (2) Joe is in a coffee room, (3) Joe is back in his office. In our language, assuming Joe's office is room 220 and CRoom contains a list of all coffee room locations, this query is:

$$q_{\text{JoeCoffee}} = \text{At}(\underline{\text{'Joe'}}, \text{'220'}); (\sigma_{\text{CRoom}(l)}\text{At}(\underline{\text{'Joe'}}, l)); \text{At}(\underline{\text{'Joe'}}, \text{'220'})$$

A more sophisticated query: "Tell me if *anyone* goes to the Coffee Room without stopping at someone else's office" can also be expressed as a sequence of multiple events: (1) a person is in his or her office, (2) the person is at a sequence of locations in the hallway (without stopping in anyone else's office), and (3) the person is in a coffee room. In our language, this query is:

$$\begin{aligned} q_{\text{AnyCoffee}} =& \sigma_{\theta_1}(\ \text{At}(\underline{p}, l_1); \text{At}(\underline{p}, l_2)^+\langle\{p\}, \theta_2\rangle; \text{At}(\underline{p}, l_3)) \\ & \text{where } \theta_1 = \mathtt{Person}(p) \text{ AND } \mathtt{Office}(p, l_1) \text{ AND } \mathtt{CRoom}(l_3) \\ & \text{and } \theta_2 = \mathtt{Hall}(l_2) \end{aligned}$$

A query returns an event with one attribute for each *free variable* and a timestamp, T; for a single subgoal g each variable in g is free, for Kleene Plus $g^+\langle V, \theta\rangle$ only the variables in V are free. For a sequence, $q = q_1; g$, the free variables are the union of the free variables in q_1 and g. We write **var**(θ), **var**(g), **var**(q) to denote the set of free variables in a condition, a subgoal, or a query. The formal semantic is in Fig. 2.

Most constructs have a standard semantics, which we informally explain below. We call a variable **shared** if it occurs in more than one subgoal or is shared in a Kleene plus. If a variable is shared it must be bound to the same value–hence it expresses a join, as in datalog (see *e.g.* the variable p denoting the person getting coffee in $q_{\text{AnyCoffee}}$). A sequence query $q; bq$ pairs the events returned by q with their immediate successors among events returned by bq. The parameterized Kleene plus $q^+\langle V, \theta_2\rangle$ is a union of a series of sequence operations, where the variables in V are shared and the condition θ_2 is applied at the end. For example, consider $\text{At}(\underline{p}, l)^+\langle\{p\}, \mathtt{Hallway}(l)\rangle$. Here the variable p is shared, while l is not, so the query asks for the same person but possibly at different locations – all of which must be hallways. On the other hand, in $\text{At}(\underline{p}, l)^+\langle\{p, l\}, \mathtt{Hallway}(l)\rangle$ both variables are shared, which means that we require the same person to be at the same location which must be a hallway in each unfolding. Bindings to the shared variables are returned by the queries, while the others are not: $\text{At}(\underline{p}, l)^+\langle\{p\}, \mathtt{Hallway}(l)\rangle$ returns a set of events with attributes p and T only.

Finally, we say that a query q is **satisfied** at time t and write $W \models q@t$, if there exists an event e with timestamp $T = t$ that satisfies q. Formally, there exists some e such that $e \in [\![q]\!]_W$ and $e[\mathtt{T}] = t$. In Lahar, for a query q, we returns the timesteps at which q is satisfied without the events.

2.3 Probabilistic Event Databases

Let $A_1, \ldots, A_k$ be k value attributes, where A_i takes values in some domain D_i. Let $\bar{D} = D_1 \times \ldots \times D_k$, and $\bar{D}_\perp = \bar{D} \cup \{\perp\}$. A *partial random variable* over $A_1, \ldots, A_k$ is a function $p : \bar{D}_\perp \to [0, 1]$ s.t. $\sum_{d \in \bar{D}_\perp} p(d) = 1$.

A **probabilistic event**, e, consists of a key (ID), a timestamp (T) and a partial random variable over its value attributes. For an illustration, let e be the event "Joe got coffee at 10:05am either in room 326 with probability 0.2, or in room 327 with probability 0.7". The event key (Joe) and the event timestamp (10:05am) are known, while the value attribute (room) is given as a random variable. We write:

$$\mathbf{P}[e = \text{'Room326'}] = 0.2,\ \mathbf{P}[e = \text{'Room327'}] = 0.7$$

Notice that with probability 0.1 Joe did not have coffee at all at 10:05am, in which case the room is $\perp$, which we write as

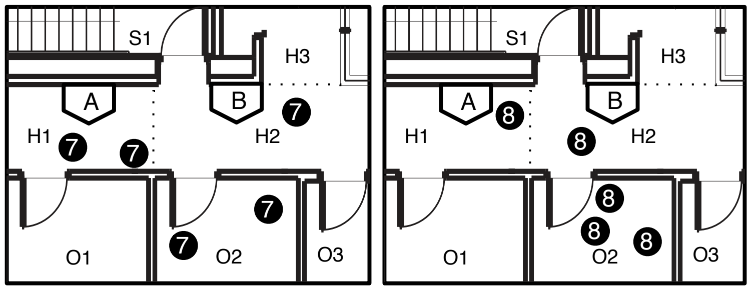
Let q be a query and W be a world. The semantics of q on W, $[\![q]\!]_W$ is a set of events e, with attributes for each shared variable and T, defined as follows:

$$\begin{aligned} [\![g]\!]_W &\overset{\text{def}}{=} \{e \mid e \in W \text{ AND } e \text{ matches } g\} \\ [\![\sigma_\theta(q_1)]\!]_W &\overset{\text{def}}{=} \{e \mid e \in [\![q_1]\!]_W \text{ and } e[\mathbf{var}(\theta)] \in \theta\} \\ [\![q_1; bq]\!]_W &\overset{\text{def}}{=} \{e \mid e_1 \in [\![q_1]\!]_W \text{ and } e_1[\mathbf{var}(q_1)] = e[\mathbf{var}(q_1)] \\ & \quad e_2 \in \mathcal{S}(e_1, bq) \text{ and} \\ & \quad e_2[\mathtt{T}] = \min_{e' \in \mathcal{S}(e_1, bq)} e'[\mathtt{T}] \text{ and } e[T] = e_2[T]\} \\ [\![q_1; \sigma_{\theta_1}(g)^+\{V, \theta_2\}]\!]_W &\overset{\text{def}}{=} [\![\sigma_{\theta_2}(q_1; \sigma_{\theta_1}(g))]\!]_W \\ & \quad \cup [\![\sigma_{\theta_2}(\mathrm{F}_{\bar{V}}\left[\sigma_{\theta_2}(q_1; \sigma_\theta(g))\right]; \sigma_\theta(g))]\!]_W \\ & \quad \cup \ldots \end{aligned}$$

where:

$$\begin{aligned} \mathcal{S}(e_1, bq) &\overset{\text{def}}{=} \{e' \mid e' \in [\![bq]\!]_W, e'[\mathbf{var}(bq)] = e_1[\mathbf{var}(bq)], \\ & \quad \text{and } e'[\mathtt{T}] > e_1[\mathtt{T}]\} \text{ (the set of events that come after } e_1) \\ \bar{V} &\overset{\text{def}}{=} \mathbf{var}(g) - V \\ \mathrm{F}_{\bar{V}}[e] &\overset{\text{def}}{=} \text{denotes substituting each variable in } \bar{V} \text{ with a fresh one} \end{aligned}$$

Figure 2: Semantics of Lahar's query language.



(a) (b)

	T	tag	loc	p
$e^{(7)}$	7	Joe	H1	0.4
			H2	0.2
			O2	0.4
	7	Sue	O2	0.6
	⋮	⋮	⋮	⋮
$e^{(8)}$	8	Joe	O2	0.6
	⋮	⋮	⋮	⋮

(c)

	tag	T	d′	d	p
$E^{(7)}$	Joe	7	O2	O2	0.7
				H2	0.4
				H3	0.1
	Joe	7	H2	O2	0.2
	⋮	⋮	⋮	⋮	⋮
	Joe	7	H3	O2	0.1
	⋮	⋮	⋮	⋮	⋮
	Sue	7	O2	O2	0.1
	⋮	⋮	⋮	⋮	⋮
	Joe	8	O2	O2	0.6
	⋮	⋮	⋮	⋮	⋮

(d)

Figure 3: (a) A visual representation of a particle filter; a particle is a guess about Joe's true location at time 7**. (b) The distribution over Joe's location at** $t = 8$**, after Joe is not detected by any readers. The system has concluded it is likely that Joe is in his office,** $O2$**, with probability** 0.6**. (c) A representation of an event relation,** At**. One stream is for Joe, highlighted as** $e^{(7)}$ **and** $e^{(8)}$**. This stream is interleaved with the stream tracking Sue's location. (d) Correlation in the** At **relation (the CPT). In particular,** $E^{(7)}$ **for Joe's stream shows the conditional probability table for times** 7 **and** 8**. The shaded rows highlight the distribution on his location at time** 8 **given that he was in his office at time** 7**. For example, given that Joe was in his office at time** 7**, he is likely to remain in his office at time** 8**, because** $E^{(7)}(O2, O2) = 0.7$ **and less likley to go to the hallway,** $E^{(7)}(H2, O2) = 0.2$**.**

$\mathbf{P}[e = \perp] = 0.1$. Events with the same key are correlated. For example the event "Joe is having coffee in Room 326 at 9:50am" is positively correlated with the event "Joe is having coffee in Room 326 at 9:55am" and negatively correlated with "Joe is having coffee in Room 327 at 9:50am" (because the room differs).

The value $\perp$ denotes the absence of an event: For example, if Joe did not get coffee one morning then all his GetCoffee events for that time will have the value $\perp$.

Let the sequence $\bar{e} = (e^{(1)}, e^{(2)}, \dots, e^{(t)}, \dots)$ be a stream of probabilistic events. We abbreviate $\mathbf{P}[e^{(t+1)} = d^{(t+1)} \mid e^{(t)} = d^{(t)}]$ with $\mathbf{P}[e^{(t+1)} | e^{(t)}]$. Then $\bar{e}$ is **Markovian** if:

$$\mathbf{P}[e^{(t+1)} | e^{(1)}, \dots, e^{(t)}] \quad = \quad \mathbf{P}[e^{(t+1)} | e^{(t)}]$$

In this case, the probability of a particular sequence of values $\bar{d} = (d^{(1)}, d^{(2)}, \dots, d^{(t)})$ is given by Bayes' rule:

$$\mu(\bar{d}) \stackrel{\text{def}}{=} \mathbf{P}[e^{(1)} = d^{(1)}] \prod_{i=2,\dots,t} \mathbf{P}[e^{(i)} = d^{(i)} | e^{(i-1)} = d^{(i-1)}] \qquad (1)$$

We assume that a single stream can have Markovian correlations across time, but that distinct streams are independent.

A **probabilistic event database**, $\mathcal{W}$, consists of a set of probabilistic event streams and, optionally, standard relations. We store and process probabilistic event databases in a relational system, as is done with deterministic event databases. Note that a Markovian stream is fully specified by the **conditional probability table** (CPT), $E^{(t)}(d', d) = \mathbf{P}[e^{(t+1)} = d' \mid e^{(t)} = d]$, for $d, d' \in \bar{D}_\perp$ and an initial marginal distribution $\mathbf{P}[e^{(0)}]$. Thus a probabilistic stream with value attributes $A_1, \dots, A_k$ can be stored in a relation with schema $E(\underline{\mathtt{ID}}, \mathtt{T}, A'_1, \dots, A'_k, A_1, \dots, A_k, P)$. An example of this encoding is shown in Fig. 3(d) (we further discuss this figure below). A particular case is that of an **independent stream**, where $E^{(t)}(d', d) = \mathbf{P}^{(t)}[d']$, in which case the schema simplifies to $E(\underline{\mathtt{ID}}, \mathtt{T}, A'_1, \dots, A'_k, P)$.

Finally, we give the formal semantics of a query over a probabilistic event database $\mathcal{W}$. Consider any world W. Let $\bar{d} = (d^{(1)}, \dots, d^{(t)})$ be a subset of events in W with the same key[2]: its probability is given by Eq.(1). Then the probability of the world W is $\mu(W) = \prod_{\bar{d} \in W} \mu(\bar{d})$.

D 2.3. *Let $\mathcal{W}$ be a probabilistic event database and q be a query. The* ***value*** *of q denoted $\mu(q)$ is defined as:*

$$\mu(q) \stackrel{\text{def}}{=} \sum_{W : W \models q} \mu(W)$$

[2]We pad with $\perp$ at all timestamps where the key is missing.

Our goal is to process event queries on the most up-to-date data in a probabilistic event database. Formally, for the remainder of the paper we study the *event query evaluation problem*:

Event Query Evaluation Problem: Given an event query q and any probabilistic event database $\mathcal{W}$ at time t, the event query evaluation problem is to calculate the probability that q is true at time t, *i.e.* $\mu(q@t)$.

2.4 Generating Event Streams

A probabilistic event stream can be obtained by performing inference on an HMM. In our motivating application, the input to Lahar is the result of inference on an HMM that determines the marginal distribution of the location of each RFID tag. In our experiments, our input is generated by a popular, sample-based inference technique called particle filtering [2]. Two examples of sampled distributions produced by the particle filter are shown in Fig. 3(a) and (b). Each particle represents a guess about Joe's location. The particle filter predicts the location of each particle at the next time step based on its current location, but independently from the location of any other particle. It then resamples the particles, choosing with a higher probability those particles that are more consistent with the sensor readings. For example, if a sighting occurs at antenna A, then particles near antenna A are more likely to be resampled. To compute a marginal probability distribution such as in Fig. 3(c), we simply count the number of particles in each location and divide by the total.

Lahar is designed for two different scenarios: a *near real-time scenario*, or simply real-time, where we immediately consume the output of inference, and an *archived scenario*, where we are able to preprocess the data. The real-time scenario models the situation when our system does not have access to correlation information and so must assume that each tuple is independent. By contrast, the archived scenario assumes correlation information is available. Specifically, we apply Bayesian smoothing [26, 31] which not only reduces the noise in the marginal distributions, but also produces consistent Markovian correlations between adjacent timesteps, which are captured in CPTs as in Fig. 3(d). In the archived scenario, Lahar uses these CPTs to extract higher-quality events. Hence, in the real-time scenario, Lahar operates on independent streams while in the archived case, it processes Markovian streams.

3. PROCESSING EVENT QUERIES

We now turn to the problem of evaluating the probability of a query q on a probabilistic event database $\mathcal{W}$. We describe four algorithms, applicable to four increasingly complex classes of queries: Regular, Extended Regular, Safe, and General.

Regular Queries We first consider regular queries, which are similar to standard regular expressions.

D 3.1 (R Q). *A predicate θ is **local** in q if $\mathbf{var}(\theta) \subseteq \mathbf{var}(g)$ for some subgoal g in q. A query q is **regular** if every predicate is local and there are no shared variables, i.e. for any distinct subgoals g, g' we have $\mathbf{var}(g) \cap \mathbf{var}(g') = \emptyset$ and Kleene pluses do not share (or export) variables.*

Example 3.2 As a simple illustration, the Regular Query below checks whether Joe went from 'a' to 'c' by going only through the `Hallway`:

$$q_{Joe,hall} = \text{At}(\underline{\text{'Joe'}}, \text{'a'}); \text{At}(\underline{\text{'Joe'}}, l)^{+}\langle\emptyset, \texttt{Hallway}(l)\rangle; \text{At}(\underline{\text{'Joe'}}, \text{'c'})$$

In Sec. 3.1 we describe an algorithm for Regular Queries. Its complexity is given below:

T 3.3. *Fix a Regular Query q. Then q can be evaluated on a probabilistic event database $\mathcal{W}$ in time $O(|\mathcal{W}|)$ and space $O(1)$. Further, we can incrementally compute q with each step requiring time $O(1)$.*

Thus Regular Queries can be evaluated in streaming fashion.

Extended Regular Queries Extended Regular Queries allow some projections and joins in addition to regular expressions.

D 3.4. *A query q is **syntactically independent** on x if (a) x is shared among all subgoals, (b) x is in a key position in every subgoal, and, (c) if g, g' are subgoals of the same stream type, then there must exist a position i in the key such that x occurs at position i in both g and g'.*

D 3.5 (E R Q). *A query q is **extended regular** if all predicates are local and for any variable x that appears in more than one subgoal (or is shared by a Kleene plus), q is syntactically independent on x.*

This definition implies that if we substitute distinct constants c_1, c_2 for x, then no tuple can unify with both $q\{x \to c_1\}$ and $q\{x \to c_2\}$.

Example 3.6 This Extended Regular Query asks for an alert when *any* person goes from 'a' to 'c' using `Hallway`s in between.

$$q_{hall} = \sigma_{\texttt{Person}(x)}(\text{At}(\underline{x}, \text{'a'}); \text{At}(\underline{x}, l_2)^{+}\langle\{x\}, \texttt{Hallway}(l_2)\rangle; \text{At}(\underline{x}, \text{'c'}))$$

Sec. 3.2 discusses the algorithm for Extended Regular Queries; its complexity is:

T 3.7. *Let q be an Extended Regular Query. Then q can be evaluated on a probabilistic event database $\mathcal{W}$ in time $O(|\mathcal{W}|)$ and in space $O(m)$, where m is the total number of distinct keys in $\mathcal{W}$. Further, we can incrementally compute Extended Regular Queries, with each step of the algorithm requiring time $O(m)$.*

Thus Extended Regular Queries can also be evaluated in streaming fashion. Their memory requirements depend on the number of distinct key values in the streams, *e.g.* the number of people and objects that the system is tracking. However, this number is independent of the length of the streams.

Safe Queries The third class of queries we consider are Safe Queries.

D 3.8 (S Q). *Fix a query q where all predicates are local. Given a variable x, let q' be the smallest subquery in q that contains all occurrences of x, then we say x is **grounded** if q' is syntactically independent on x. A query q is **safe** if every predicate is local and any variable that appears in more than one subgoal (or is shared by a Kleene plus) is grounded in q.*

Recall that a subquery q' is a prefix of q. Thus, in Def. 3.8, q' is the shortest prefix of q that contains all occurrences fo x. In particular, x must occur in each subgoal of this prefix, since q' is syntactically independent on x.

Example 3.9 The following Safe Query checks whether a person x went to a talk: It first checks that the person and a laptop were in an office, then that the person and the laptop were together until the person entered a lecture room:

$$q_{talk} = \sigma_{\theta}(\text{Carries}(\underline{x, y}, z); (\text{Carries}(\underline{x, y}, -))^{+}\langle\{x, y\}, \rangle; \text{At}(\underline{x}, u))$$

where $\theta(x, y, z, u) = \texttt{Person}(x)$, $\texttt{Laptop}(y)$, $\texttt{Office}(z)$, $\texttt{LectureRoom}(u)$.

Sec. 3.3 discusses an algorithm for evaluating Safe Queries. Its complexity is:

T 3.10. *Let q be a Safe Query. Then q can be evaluated on a probabilistic event database $\mathcal{W}$ in time $O(|\mathcal{W}|^2)$ and space $O(T^2)$, where T is the number of distinct timestamps in $\mathcal{W}$.*

Thus while Safe Queries can be computed efficiently on a stored probabilistic event database, they cannot be computed in streaming fashion because the history that we need to maintain increases with the size of the streams. We describe in Sec. 3.3 extensions and variations of Safe Queries, whose definitions are more involved, but which have the same complexity.

Finally, in Sec. 3.4, we show that even minor extensions of safe plans are provably hard to evaluate, *e.g.* even a single non-local predicate is intractable. For these most general queries we describe a general-purpose evaluation algorithm in Sec 3.5.

3.1 Regular Queries

Despite their names, Regular Queries have a more subtle semantics than regular expressions. Consider the following example:

Example 3.11 Here, q_f and q_s are seemingly equivalent Regular Queries, but they are not:

$$q_f = R(\underline{a}); R(\underline{b}) \text{ and } q_s = \sigma_{y=\text{'b'}}(R(\underline{a}); R(\underline{y}))$$

Intuitively, the two queries differ only on the order in which they apply the condition y = 'b': q_f uses it to filter the stream, while q_s applies the condition after first selecting a successor for $R(a)$. More concretely, consider the input sequence $I = \{R(a, 1); R(c, 2); R(b, 3)\}$, *i.e.* an $R(a)$ event at time 1, followed by an $R(c)$ event at time 2, followed by an $R(b)$ event at time 3. At time 1, q_f sees the tuple $R(a)$, and now looks for a successor, but only among $R(b)$ tuples; it finds one at time 3 and so is true at $t = 3$. In contrast, q_s also sees the $R(a)$ tuple at time 1, but now looks for its successor among *all* tuples of the form $R(x)$ – there is only one successor at time $t = 2$, $R(c)$, which does not satisfy the predicate $x = b$. Thus, q_f is not satisfied by this input stream.

In spite of these subtleties, we show that Regular Queries can be translated into standard regular expressions by choosing an appropriate input alphabet and translating the input probabilistic event database into this alphabet. Our technique also encodes the action of the NFA as a Markov Chain, which allows us to keep track of probabilities using matrix multiplication.

3.1.1 Translation to Regular Expressions

To reduce the number of cases for our translation, we first push down selections as far as possible by repeatedly applying the following two identities to the query: first, $\sigma_\theta(q_1; q_2) = \sigma_\theta(q_1); q_2$ if $\mathbf{var}(\theta) \subseteq \mathbf{var}(q_1)$ and then $\sigma_{\theta_1}(\sigma_{\theta_2}(q_1)) = \sigma_{\theta'}(q_1)$, informally we take $\theta' = \theta_1 \wedge \theta_2$. At the end of this process, there are two cases, either: (1) a selection applies directly to a subgoal $\sigma_\theta(g)$, for example in $\sigma_{y=b}(R(y))$, which we might as well replace with $R(b)$, or (2) a selection applies to the last subgoal in its child sequence, *e.g.* in $\sigma_\theta(\ldots; g)$ we have $\mathbf{var}(\theta) \subseteq \mathbf{var}(g)$. Both q_f and q_s are unchanged by these rules.

Our translation consists of four steps: (I) defining a set of symbols L_q, on which our automaton operates (II) translating an event database W (without probabilities) into a sequence of subsets of L_q: that is, let $\Sigma = \mathcal{P}(L_q)$, the power set of L_q, then any event database W is translated into a word in Σ^*. (III) translating the query q into a regular expression E_q over Σ, and (IV) recovering the Markov chain over Σ that is induced by the probabilistic event database $\mathcal{W}$. We show these steps next.

(I) Let $g_1, \ldots, g_n$ be a left-to-right enumeration of the subgoals of q. Since q is regular, each predicate in a query q is associated with a single subgoal. Without loss of generality, we also may assume that every subgoal g_i is associated with exactly one predicate σ_i. For example, q_f introduces a trivial selection, that is, q_f becomes $\sigma_{\mathbf{true}}(R(\underline{a}); R(\underline{b}))$. Further, we may also assume base queries do not contain selections. We then define $L_q \stackrel{\text{def}}{=} \{m_1, \ldots, m_n\} \cup \{a_1, \ldots a_n\}$. Here, for each $i = 1, \ldots, n$, m_i and a_i are simply distinct constants; we give them a semantics below. Finally, since multiple symbols can occur simultaneously, we define the language of the automaton, Σ, to be $\Sigma = \mathcal{P}(L_q)$.

(II) Let W be an event database: we translate this into a sequence $\bar{S}_W^{(T)} = S^{(1)}S^{(2)} \ldots S^{(T)} \in \Sigma^*$. For each t, let $W^{(t)}$ be the set of events with timestamp t. Then the set $S^{(t)}$ is derived from $W^{(t)}$ as follows. If $W^{(t)}$ contains an event that *matches* (or unifies with) a subgoal g_i, then we insert m_i into $S^{(t)}$; if it matches both the subgoal g_i and is *accepted* by its predicate σ_i, then we insert both m_i and a_i into $S^{(t)}$. In Ex. 3.11, $R(c, 2)$ is an event for q_s for which an m_2 is produced without a corresponding a_2, since c does not satisfy $\sigma_2 = \sigma_{y=b}$.

(III) We rewrite the input query into regular expressions over Σ with the following syntax:

$$E = P \mid (E, E) \mid E^+ \mid E^*$$

where P is an atomic predicate on the alphabet Σ of the following form: P is either a set S, or the negation of a set $\neg S$, where $S \subseteq L_q$. In the first P matches any input in Σ that is a superset of S, and in the second it matches any input that is disjoint from S. The translation of a query q into a regular expression E_q is given by:

q	E_q, its translation
g_1	$\{a_1\}$
$\sigma_i(q'; g_i)$	$E_{q'}, (\neg\{m_i, a_i\})^*, a_i$
$q'; g_i^+\langle\emptyset, \theta\rangle$	$E_{q'}, ((\neg\{m_i, a_i\})^*, a_i)^+$

Example 3.12 Returning to our previous example, we see that the regular expression translations of q_f and q_s are the same:

$$E_{q_f} = E_{q_s} = \{a_1\}, (\neg\{m_2, a_2\})^*, \{a_2\}$$

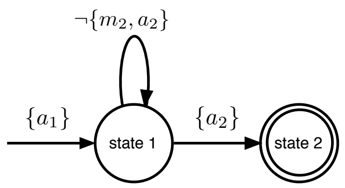
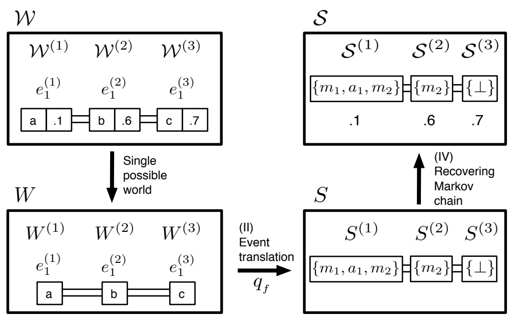


Figure 4: A sample automaton corresponding to both q_f or q_s of Ex. 3.11. Implicitly, the automaton is always in the start state, hence it is not drawn. State 2 is a final state.



$\mathcal{W}$
$\mathcal{W}^{(1)}$ $\mathcal{W}^{(2)}$ $\mathcal{W}^{(3)}$
$e_1^{(1)}$ $e_1^{(2)}$ $e_1^{(3)}$
a .1 b .6 c .7
$\mathcal{S}$
$\mathcal{S}^{(1)}$ $\mathcal{S}^{(2)}$ $\mathcal{S}^{(3)}$
$\{m_1, a_1, m_2\}$ $\{m_2\}$ $\{\perp\}$
.1 .6 .7
Single possible world
(IV) Recovering Markov chain
W
$W^{(1)}$ $W^{(2)}$ $W^{(3)}$
$e_1^{(1)}$ $e_1^{(2)}$ $e_1^{(3)}$
a b c
(II) Event translation
q_f
S
$S^{(1)}$ $S^{(2)}$ $S^{(3)}$
$\{m_1, a_1, m_2\}$ $\{m_2\}$ $\{\perp\}$

Figure 5: A probabilistic event database $\mathcal{W}$ represents many possible worlds. Here, $\mathcal{W}$ consists of only a single stream (correlations are not pictured). In step (II), a single possible world $\bar{W}^{(t)} = W^{(1)} \ldots W^{(t)}$ maps to a word in Σ^*, denoted $\bar{S}^t = S^{(1)}, \ldots, S^{(t)}$. In step (IV), we recover the distribution of S, which is denoted $\bar{\mathcal{S}}^{(t)} = \mathcal{S}^{(1)} \ldots, \mathcal{S}^{(t)}$.

A sample automaton that captures this language is illustrated in Fig. 4. However, Lahar translates the input from Ex. 3.11 differently for each query:

Time	Input	$S^{(t)}$ for q_f	$S^{(t)}$ for q_s
1	$R(a)$	$\{m_1, a_1\}$	$\{m_1, a_1, m_2\}$
2	$R(c)$	$\emptyset$	$\{m_2\}$
3	$R(b)$	$\{m_2, a_2\}$	$\{m_2, a_2\}$

This translation correctly preserve the semantics: q_f is true because the input string is $\{m_1, a_1\}$, $\emptyset$, $\{m_2, a_2\}$ which matches the regular expressions $\{b_1\}$, $\neg\{m_2, a_2\}^*$, and $\{a_2\}$ respectively. However, q_s is false, because here the second symbol is $\{m_2\}$ which does not match $\neg\{m_2, a_2\}^*$ nor $\{a_2\}$. Said another way, if automaton is in State 1 (Fig. 4), then on reading $\{m_2\}$, it must move. However, it can take neither the self-loop, annotated with $\neg\{m_2, a_2\}$, nor the outgoing edge, annotated with $\{a_2\}$ and so must fail to accept.

The final step in our translation is to prepend a wildcard star (*e.g.* $.*$) at the beginning of E_q, since event queries may start at any time. The translation is correct in the following sense:

P 3.13. *$W \models q@T$ iff E_q accepts $\bar{S}_W^{(T)}$.*

(IV) Finally, we show how to recover the Markov chain that we use to evaluate the regular expression E_q. The distribution of this chain is induced by the probabilistic event database $\mathcal{W}$. Formally, we do this by adding the sets $S^{(t)}$ to the events in $\mathcal{W}$; the regular expression uses only the sets $S^{(t)}$, while the presence of all other events from $\mathcal{W}$ ensures that the chain is Markovian. This process is illustrated in Fig. 5.

Formally, we define a sequence of random variables $\bar{\mathcal{S}}^{(T)} = \mathcal{S}^{(1)}\mathcal{S}^{(2)} \ldots \mathcal{S}^{(T)}$, which capture the distribution of $S^{(1)} \ldots, S^{(T)}$ in Fig. 5. Recall that $\mathcal{W}$ is a collection of n-Markov chains,

$e_1, e_2, \ldots, e_n$, where each e_i is a probabilistic stream. Let $\mathcal{W}^{(t)}$ be the set of all probabilistic events with timestamps t in all Markov chains in $\mathcal{W}$. We view $\mathcal{W}^{(t)}$ as a random variable whose values are sets of events, $W^{(t)}$. Formally, let $S^{(t)}$ be the set of m_i's and a_i's constructed via (II) above; observe that $S^{(t)}$ is only a function of $W^{(t)}$. Then we define $\mathcal{S}^{(t)}$ to be a random variable that takes the value $W^{(t)} \cup S^{(t)}$ with probability $\mathbf{P}[W^{(t)}] = \prod_{e^{(t)} \in W^{(t)}} \mathbf{P}[e^{(t)}]$. This is a Markov chain, because the random variables in the set $\mathcal{W}^{(t+1)}$ depend only on those in the set $\mathcal{W}^{(t)}$. The conditional probability table for the sequence $\mathcal{S}^{(t)}$ is the product of the CPTs for all Markov chains in $\mathcal{W}$. Note that we cannot define $\mathcal{S}^{(t)}$ to be just $S^{(t)}$ because the probability of a transition $\mathcal{S}^{(t)}$ to $\mathcal{S}^{(t+1)}$ depends not only on $S^{(t)}$, but also on the hidden state $W^{(t)}$. Thus, we simply include the hidden state in $W^{(t)}$ hidden state in $\mathcal{S}^{(t)}$.

In the real-time scenario, where all event streams are independent, the definition of an individual Markov Chain, $e^{(1)}, \ldots, e^{(t)}$, is simpler: The state at time $t+1$ does not depend on the state at time t. The remainder of the construction is unchanged.

3.1.2 Regular Expressions on Markov Chains

Let E be a regular expression over an alphabet Σ, and δ be the transition function for the corresponding automaton. Consider T random variables over Σ, $\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(T)}$, and suppose they form a Markov chain:

$$\mathbf{P}[\mathcal{S}^{(t)} | \mathcal{S}^{(1)} \ldots \mathcal{S}^{(t-1)}] = \mathbf{P}[\mathcal{S}^{(t)} | \mathcal{S}^{(t-1)}]$$

The Markov chain is thus completely defined by the conditional probability tables, $C^{(t)}(\sigma', \sigma) = \mathbf{P}[S^{(t)} = \sigma' | S^{(t-1)} = \sigma]$, for $t = 1, \ldots, T$ (we assume $S^{(0)}$ to be some fixed symbol in Σ). It defines a probability space over words of length T in Σ^*, *i.e.* each word w has some probability $\mathbf{P}[w]$ of being generated by the Markov process. We define the *probability* of the regular expression E, $\mathbf{P}[E]$, to be the probability that a randomly chosen word w satisfies E. Our goal is to compute $\mathbf{P}[E]$ from the expression E and the CPT $C^{(t)}$, $t = 1, \ldots, T$.

Let A be a deterministic automaton for the expression E, and let Q be its states. Denote $w = w^{(1)} \ldots w^{(T)}$ a random word generated by the Markov chain. For each $t = 1, \ldots, T$, we define a random variable $M^{(t)}$ with values in $\Sigma \times Q$ as follows. For each $\sigma \in \Sigma, q \in Q$ the random variable $M^{(t)}$ takes the value (σ, q) if the following event holds: "$w^{(t)} = \sigma$ and the automaton is in state q after reading the symbol $w^{(t)}$". In Ex. 3.12, suppose that event $R(a)$ at time 1 has a probability of 0.5, then the Markov Chain would be in the state $(\{\text{'State 1'}\}, \{m_1, a_1\})$ (cf. Fig. 4). Then $M^{(1)}, M^{(2)}, \ldots, M^{(T)}$ forms a Markov chain, and we can compute their probabilities iteratively as follows:

$$\mathbf{P}[M^{(t)} = (\sigma', q')] = \sum_{\sigma \in \Sigma, q \in Q : \delta(q, \sigma') = q'} C^{(t)}(\sigma', \sigma) \cdot \mathbf{P}[M^{(t-1)} = (\sigma, q)]$$

In Lahar, this equation is implemented straightforwardly: the regular expression is first compiled to an NFA and we maintain the distribution of states of the automaton explicitly. This means that our evaluation is essentially matrix multiplication.

In the real-time scenario, we can improve the algorithm using the fact that the next letter seen by the automaton is independent of the previously seen letters. In this case, $M^{(t)}$ can be defined by a simpler event "the automaton is in state σ", which does not mention the previously seen letter. This results in a smaller automaton.

3.2 Extended Regular Queries

Recall the Extended Regular Query, q_{hall} from Example 3.6; to answer it we need to track each tagged person independently. Consider a naïve method for computing such queries: substitute each possible constant for x, create a large automaton, and use the algorithm from Sec. 3.1 to process it. This approach is untenable because it requires space that is *exponential* in the number of possible constants (people), since the distribution is now over all possible subsets of people. In this section we give an efficient algorithm for queries involving projection and join that avoids computing this large automaton directly. We illustrate the method by example:

Example 3.14 $q\{x \to d\}$ denotes the result of substituting all occurrences of x with d in q. For example, let $q_{joe} = q_{hall}\{x \to \text{'Joe'}\}$,

$$q_{Joe} = \sigma_{\texttt{Person('Joe')}}(At(\underline{\text{'Joe'}}, \text{'a'}); At(\underline{\text{'Joe'}}, l_2)^{+}\langle \emptyset, \texttt{Hallway}(l_2)\rangle; At(\underline{\text{'Joe'}}, \text{'c'}))$$

q_{Joe} is a regular query, so we can compute its probability using the Markov chain approach from the previous section. Our goal is to compute the probability that the query is true for *any* value of x, not just for a single binding. For q_{hall}, we make a critical observation: If we bind x to different constants, say d, d', then $q_d = q\{x \to d\}$ and $q_{d'} = q\{x \to d'\}$ use disjoint sets of tuples. The events contributing to q_d and $q_{d'}$ are therefore *independent*. This is significant because it implies that we can now efficiently compute the probability of their disjunction, which is $\mathbf{P}[q_{hall}]$.

Formally, the key insight from Ex. 3.14 is that q_{hall} is *syntactically independent* on x (Def. 3.4). This implies a straightforward algorithm for computing $\mathbf{P}[q]$ for any Extended Regular q: Fix an enumeration of all possible bindings for all x that occur in more than one subgoal, *e.g.* $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n$. For $i = 1, \ldots, n$, let $p_i^{(t)}$ be the probability that the Markov chain corresponding to the regular query $q_{\boldsymbol{d}_i}$ is true at time t. Then the probability that an Extended Regular Query q is true is exactly $1 - \prod_{i=1}^{n}(1 - p_i^{(t)})$.

Since each chain can be computed in $O(1)$ space and there are m chains, this approach requires $O(m)$ space and can be computed incrementally using $O(m)$ multiplications, as stated in Thm. 3.7.

Completeness Checking whether a query is syntactically independent is easy to implement efficiently inside a compiler. However, given that a similar syntactic condition is known to fail for conjunctive relational queries [29, 36] it is natural to wonder whether our check is complete. Surprisingly, for event queries, we can show that our simple check is complete, we include the proof in our full version [35].

3.3 Safe Queries

Recall q_{talk} from Ex. 3.9. To handle this query we need to check that the same person was carrying some laptop on her way to the talk, but then we do not care whether her laptop makes it to the lecture room. This query is not Extended Regular because the variable y appears in the first two subgoals, but not the last. Intuitively, this query requires that we are able to interleave a projection (we don't care which laptop) and a sequencing operation. To handle these more complicated queries we adopt a different strategy from previous sections: We use an algebra for probabilistic streams, which is inspired by the *Probabilistic Relational Algebra* ($\mathcal{PRA}$) [15]. The goal of the algebra is to compute the probability of a query efficiently, using simple operators that add and multiply probabilities.

3.3.1 Probabilistic Stream Algebra

Our algorithm uses query plans, called **safe plans**, which consist of four algebraic operators that we define below. A safe plan is a left-deep plan where the leftmost leaf is the set of regular expression queries, where all shared variables have been substituted with constants. In a safe plan, every right child is a simple *base query*,

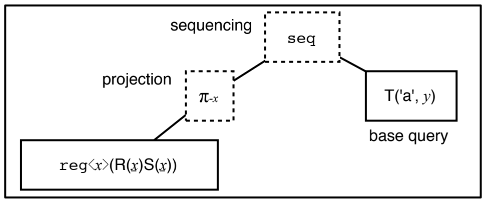


Figure 6: An example *safe plan* for $q = R(\underline{x}); S(\underline{x}); T(\underline{\textbf{'a'}, y})$.

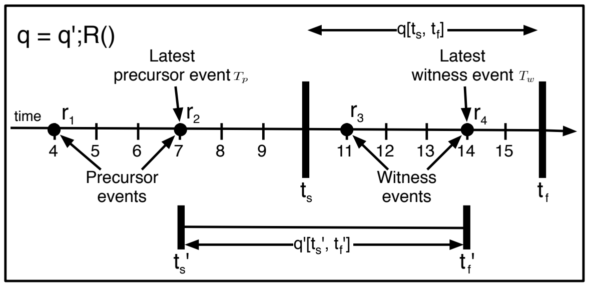


Figure 7: A possible world for R, annotated with information used in computing $\texttt{seq}(P', bq)$ where P' is a safe plan for q' and $q = q'; bq$. Our goal is to compute $\mathbf{P}\Big[\ q[t_s, t_f]\ \Big]$ in terms of $\mathbf{P}\Big[\ q'[t'_s, t'_f]\ \Big]$.

which is just a set of streams relation of the same type on with simple selections and projections. An example safe plan is shown in Fig. 6. We show how it is constructed in Sec. 3.3.2.

D 3.15. *A **safe plan** is a left-linear tree of algebraic operators. The leftmost leaf of a safe plan is a regular expression operator, denoted* $\texttt{reg}\langle V_{\texttt{reg}}\rangle(q)$*, where* $V_{\texttt{reg}}$ *is a set of variables and q is a query. If we substitute any constants for* $V_{\texttt{reg}}$ *in q, say* $\boldsymbol{d}$*, then* $q\{V_{reg} \to \boldsymbol{d}\}$ *is regular.* $V_{\texttt{reg}}$ *is all variables shared by q with the rest of the query (in Fig. 6,* $V_{\texttt{reg}} = \{x\}$*).*

Inductively, if P_1 *is a plan and bq a base query, then all of the following are plans:* $\sigma_\theta(P_1)$ *(selection),* $\texttt{seq}(P_1; bq)$ *(sequence), and* $\pi_{-x}(P_1)$ *(projection) which intuitively removes the variable x.*

To determine that q is true at time t, we need to understand when subqueries q' of q are true not just at a single point, but during an interval $[t_s, t_f]$. By during an interval, we mean that there exists *some* point t, such that $q'@t$. This interval semantic is necessary to get the induction off the ground. We denote the fact that q returns e during $[t_s, t_f]$ as $q(e)[t_s, t_f]$ and define its semantics in the obvious way.

Our goal for the remainder of the section is to explain how we calculate the probability that q is satisfied during $[t_s, t_f]$, which we write as $\mathbf{P}[\ q(e)[t_s, t_f]\ ]$. Inductively, we may assume that a subplan is able to correctly compute this value for any settings of e, t_s and t_f. To simplify the discussion and notation, we assume that all attributes are part of each key and all events in any stream are independent.

Sequence We begin by illustrating the most interesting of our operators, $\texttt{seq}$ on a fixed Boolean query, $q = q'; R()$, *i.e.* there are no shared variables between q' and $R()$ (above $bq = R$). A precondition of $\texttt{seq}(q_1; g)$ is that no event can unify with both g and any subgoal of q_1. Fig. 7 shows the R events, $r_1, \ldots, r_4$, in a single possible world on an axis representing time. We use this figure to illustrate our discussion below. For q to be satisfied during $[t_s, t_f]$ at least two things must happen: First, there must be an $R()$ event in $[t_s, t_f]$. We call any such event a *witness event*, *e.g.* r_3 and r_4 in Fig. 7. Second, q' must be true at some point occurring earlier than some witness event. However, it is not enough that q' be satisfied simply before a witness event. To see this, suppose that q' is true only at time 5. Then in the possible world illustrated in Fig. 7, the successor of q' is r_2. r_2 will thus consume the q_1 event, leaving no q' event to join with r_3 and satisfy q during $[t_s, t_f]$. Hence we refer to $R()$ events in the interval $[0, t_s]$ as *precursor events*. To satisfy q, we require that q' is true after all precursor events, but also before some witness event in $[t_s, t_f]$; this is the lower interval illustrated in Fig. 7, from timesteps 7 to 14. Let T_p (resp. T_w) be the latest precursor timestamp (resp. latest witness timestamp). Let t'_s and t'_f be times such that $t'_s \leq t'_f$, our example shows that if $T_p = t'_s$, $T_w = t'_f$, and $q'[t'_s, t'_f]$ is true, then we can conclude that $q[t_s, t_f]$. In words, if q' is true in between the latest precursor and the latest witness, then q is true. This is intuitively why we need intervals. Thus we are lead to the following recursive formula:

$$\mathbf{P}[\ q[t_s, t_f]\ ] = \mathbf{P}[\exists t'_s, t'_f \text{ s.t. } T_p = t'_s, T_w = t'_f \text{ and } q'[t'_s, t'_f]\ ] \qquad (2)$$

To compute Eq. 2 efficiently we observe that it can be factored into the following form:

$$\mathbf{P}[\ q[t_s, t_f]\ ] = \sum_{t'_s, t'_f} \mathbf{P}[T_p = t'_s] \mathbf{P}[T_w = t'_f] \mathbf{P}[\ q'[t'_s, t'_f]\ ] \qquad (3)$$

To see that this equation is correct, we first observe that in any possible world there is a *unique* latest precursor time and latest witness time. Thus, the event that the pair (T_p, T_w) takes a distinct value is disjoint, which justifies the summation. Additionally, T_p and T_w are *independent* events because they draw from disjoint sets of events. Finally, all three random variables, $q'[i, j]$, T_p, and T_w, are independent because no single event can unify with more than one of them. Eq. 2 can be computed in time $O(t^2)$ where t is the number of distinct timestamps. Extending to the general case, where q' and R can share variables, is straightforward.

Regular Expression The next interesting operator is the regular expression operator which must compute the probability that a regular expression is true during an interval $[t_s, t_f]$. Our Markov chain algorithm from Sec. 3.1 only computes the probability that the query is true at a *single* point in time, instead of during *any* point during a given interval. To generalize to intervals, the key step is to break apart the computation of $\mathbf{P}[\ q(e)[i, n+1]\ ]$ into simpler components, which we accomplish using the following recursive equation, $\mathbf{P}[\ q(e)[i, n+1]\ ] =$

$$\sum_{Q_0 \subseteq Q} \mathbf{P}[q[i, n] \mid M^{(t)} = Q_0]\, \mathbf{P}[q@n+1 \mid M^{(t)} = Q_0]\, \mathbf{P}[M^{(t)} = Q_0]$$

In words, this formula asserts that the value of $q[i, n]$ and $q[n+1, n+1]$ are *conditionally independent*, given the state of the Markov chain at time n, *i.e.* $M^{(n)}$. Furthermore, both terms in the formula can be calculated efficiently: $\mathbf{P}[\ q(e)[i, n]\ ]$ is given by recursion, while $\mathbf{P}[\ q(e)[n+1, n+1] | M^{(n)}]$ can be computed with a simple modification of our previous Markov chain algorithm.

Other Operators The other operators in our algebra are essentially standard $\mathcal{PRA}$ operators. Selection does not change the value computed by the subplan if the condition $\theta(e)$ is true and returns 0 if $\theta(e)$ is false. Projection computes exactly as described in Sec. 3.2. We verify the preconditions of our operators below (Sec. 3.3.2).

Summarizing our discussion:

T 3.16. *Given a query Q, if there exists a safe plan for q then Q has an algorithm with* $O(|\mathcal{W}|T^2)$ *data complexity, where T is the number of timestamps.*

Algorithm 1 Plan(env, q) : Compute a safe plan for q.

```
Input: A list of eliminated variables env (initially $\emptyset$) and a query $q$
Output: A safe plan for $q$ (or fail)
    (* cannotUnify($q_1, q$) returns true if no event can match a goal in $q_1$
    and $g$. shared($q$) is the set of shared variables in $q$. *)
1: if shared($q$) $\subseteq$ env then (* Check if $q$ is regular *)
2:     return reg⟨env⟩($q$)
3: else if exists $x \notin$ env such that $q$ is independent on $x$ then
4:     return $\pi_{-x}$(Plan($\{x\} \cup$ env, $q$))
5: else if $q = \sigma(q_1)$ then
6:     return $\sigma$( Plan(env, $q_1$) )
7: else if $q = q_1; g$, cannotUnify($q_1, g$) and var($q_1$)$\cap$var($g$) $\subseteq$ env. then
8:     return seq( Plan(env, $q_1$), $g$)
9: else Fail
```

3.3.2 Compiling Safe Plans

In this section we give an efficient algorithm to compile *safe queries* into safe plans. The algorithm is easiest to illustrate by example. To simplify the presentation, we assume that all variables are part of the key.

Example 3.17 Recall $q = R(x); S(x); T(\text{'a'}, y)$ from Fig. 6. We now show how Lahar constructs a safe plan for q:

Initially, the first call to Alg. 1 is as **Plan**$(\emptyset, q)$. Because q is not regular (line 1) nor is q syntactically independent on any variable (line 3), **Plan** gets to line 7, the check for `seq`. q satisfies the check for `seq` because $R(\underline{x}); S(\underline{x})$ and $T(\underline{\text{'a'}, y})$ do not share variables and no event can unify with both. We now recursively compute P_1 = **Plan**$(\emptyset, R(x); S(x)$). If this succeeds then $P_1 = \texttt{seq}(P_1, T(\text{'a'}, y))$.

To compute **Plan**$(\emptyset, R(x); S(x))$, the algorithm finds that $R(x); S(x)$ is independent on x (line 3). We now recurse again to compute P_2 = **Plan**$(\{x\}, R(x); S(x)$) (line 4). If this recursion succeeds, then **Plan**$(\emptyset, R(x); S(x)) = \pi_{-x}(P_2)$.

Finally, to compute **Plan**$(\{x\}, R(x); S(x))$, we now determine that all shared variables in $R(x); S(x)$ (namely, x) are contained in env, here just $\{x\}$. Thus, we return $\texttt{reg}\langle x\rangle(R(x); S(x))$ (line 2). Tracing backwards through the stack, yields exactly the plan in Fig. 6.

3.4 Completeness

In this section, we study the completeness of our algorithms and find that many simple extensions lead immediately to intractability. We observe that even the simplest non-Safe queries are intractable. Specifically, queries containing arbitrary non-local predicates or parametrized Kleene pluses are intractable.

P 3.18. *The following two queries have $\sharp\mathcal{P}$-hard data complexity: $h_1 = \sigma_{\theta(x,y)}(R(x); S(y))$ and $h_2 = R(); S(x)^{+}\langle\{x\}, \textbf{true}\rangle$.*

Both parts of the proposition are reductions from counting the number of solutions to a monotone bipartite 2DNF formula, which is $\sharp\mathcal{P}$-Hard [33]. Thus, it makes sense to consider queries that contain only local predicates, that do not contain Kleene plus or repeated stream symbols and for which all attributes are part of the event key. We call such queries **read-once queries**. Using our syntactic definition of safety (Def. 3.8), we can immediately see that the two simplest queries, which are read-once but not *safe* are provably hard:

P 3.19. *The following two queries have $\sharp\mathcal{P}$-hard data complexity $h_3 = R(); S(x); T(x)$ and $h_4 = R(x), S(); T(x)$*

The proof for h_3 again uses using an encoding similar to our previous reduction. However, the proof that h_4 is hard requires much more work; it uses a complicated Turing reduction based on polynomial interpolation. The fact that even the simplest non-Safe Queries are intractable suggests that our algorithm is complete.

Entity	Measure
People	8
Objects	52
Locations	352
Area	≈ 10k sq.ft.
Antennas	38
Duration	71.8 mins

(a)

Data	Size (MB)	Tuples
Filtered Probs	190MB	5.2M
Smoothed Probs	190MB	5.2M
Smoothed CPTS	26G	509M
Viterbi Paths	2MB	75k

(b)

Figure 8: Real data statistics. (a) the real world represented in our experiment (b) the data streams we produced.

3.5 Generic Algorithm

Our general algorithm can process any event query, but conceptually requires many concurrent executions of the query. We recall the following simple application of naïve random sampling:

P 3.20 (*e.g.* [30]). *Fix any Boolean query q over a probabilistic event database $\mathcal{W}$. Given any $\varepsilon, \delta > 0$, there is an algorithm returning an estimate $\tilde{\rho}$ of $\rho \overset{\text{def}}{=} \mu(q)$ for any k satisfying:*

$$\mathbf{P}[|\tilde{\rho} - \rho| \leq \varepsilon] \geq 1 - \delta$$

and runs in time $O(\varepsilon^{-2} \log \frac{2}{\delta} \cdot \text{Eval}(q))$, where $\text{Eval}(q)$ is an upper bound on the time required to compute q. It is important to note that **P** *is taken over the random choices of the algorithm.*

To achieve this guarantee we can run n copies of the query. In each timestep, we count the number t of the n that are satisfied and finally divide by n to get an estimate of $\mathbf{P}[q]$. To achieve the desired confidence and precision, δ and ε, we need to take $n = O(\varepsilon^{-2} \log \frac{2}{\delta} \cdot \text{Eval}(q))$. Of course, running several parallel copies is grossly inefficient and so we use a simple technique based on bitvectors.

4. EXPERIMENTS

Our experiments verify two key claims on real and synthetic data: First, that our approach improves the quality of detected events as compared with non-probabilistic approaches. Second, that our approach is efficient, *i.e.* our prototype system can process tens of thousands to hundreds of thousands tuples per second for Regular, Extended Regular and Safe Queries.

4.1 Experimental Setup

Architecture In our current implementation, each query is run in a separate process which receives one stream from the particle filter per person, more generally per key per stream. The query processor in each process loads all predicates needed by the query.

Experimental Data Our experiments use data from a building-wide RFID deployment [40] and synthetic traces from a simulator on the same environment. The real data was generated by 8 volunteers and 52 associated objects (*e.g.* keys, coffee mugs, and laptops) as they moved throughout two floors of our building for a little over one hour (two thirty minute time periods with a ten minute break of no readings in between); this simulated the life cycle of two days in the building. Each person and object was equipped with an RFID tag. Readings from each tag were used as input to a separate particle filter. A key benefit of the particle filter is its ability to predict distributions over rooms, despite the fact that rooms in our building are not equipped with RFID readers. Figs. 8(a) and (b) summarize statistics about our experiment.

Experimental Scenarios We evaluate our approach both in the real time and archived contexts (see Sec. 2.4). For the real-time context, we compare against a standard deterministic approach that we call MLE. This approach picks the single *most likely* (highest probability) tuple at each timestep; the result is a stream of deterministic events on which we directly process the query. In the archived context we preprocess the data to compute a single most likely path through the data, also known as the *maximum a posterori estimate* (MAP). This is constructed using the Viterbi algorithm [42]. The comparisons we make in our experimental scenarios are summarized in the following table:

Scenario	Our Approach	Competitor
	Marginals/Correlations	
Real Time	Filtered/Independent	MLE
Archived	Smoothed/Markov	MAP

Implementation Details Our prototype query system is implemented in slightly more than 9000 lines of OCaml, a garbage-collected functional language, which is well-suited to the static analysis used in our algorithms. The particle filter implementation and infrastructure is written in about 4000 lines of Java code.

4.2 Quality

We first examine a single query that captures many of the phenomena observed during our study. The query asks for a person going to the coffee room: $\sigma_{\theta(l_1,l_2,l_3)}(\mathrm{At}(p,l_1);\mathrm{At}(p,l_2);At(p,l_3))$, where $\theta(l_1,l_2,l_3) = \texttt{NotRoom}(l_1), \texttt{NotRoom}(l_2), \texttt{CoffeeRoom}(l_3)$, *i.e.* the person is outside the coffee room for two consecutive timesteps and then inside the coffee room. Forty-four events match this query in the real data.

Metrics We measure quality using two independent metrics, precision and recall, and a derived measure, the F1 measure, which is defined as the harmonic mean of precision and recall. Precision is the fraction of events detected by our system that correspond to real world events, while recall is the fraction of real world events that are detected by our system. The timestamps for real world events come from annotations by our study participants and are thus imprecise. To account for this imprecision, we consider that an event e is a *match* for a real world event r, if e occurs within d seconds of r. In this section, we use $d = 30$ seconds. We discuss the impact of the d parameter in Section 4.2.2.

Lahar's approach returns event tuples annotated with probabilities; to compare our approach to deterministic approaches, we first convert sets of probabilistic events into sets of deterministic events by thresholding. In particular, we define a *threshold parameter* ρ, such that if our system returns p as the probability that a query was true at time t, we only consider that the event occurred if $p > \rho$.

4.2.1 Main Quality Experiments

Real Time Scenario Fig. 9 shows the values of the quality metrics as we vary the threshold parameter $\rho \in [0 : 0.5]$, the range where probabilities are most useful. Indeed, if one is only interested in very high-probability events, then the MLE is sufficient. Our approach offers benefits when events are uncertain; for example, our representative test query asks for sequences of three person-location events. Even if each person-location event has a probability as high as 0.75, the probability of the coffee-room event will only be $0.75^3 = 0.42$. Thus even the most likely events returned by Lahar rarely have high probabilities, *e.g.* higher than 0.8.

As expected, Fig. 9 shows that our recall decreases and precision increases with ρ. More importantly, the figure shows that for ρ in the range $[0.1, 0.5]$ our approach's benefit is dramatic: we outperform MLE in *both* precision by as much as 16 points (28% increase) *and* recall by 11 points (at most a 13% increase). A way to concisely capture this gain in both metrics is the increase in F1-measure, shown in Fig. 9(c). We outperform MLE because MLE is forced to choose a single location out of many, even in situations when there is no clearly preferred location. In contrast, our approach is able to retain the entire set of alternatives. This explains our gains in precision as well: If we consider a scenario where we miss a reading in the neighborhood of the coffee room, the MLE will likely continue to erroneously enter the coffee room, while we assign the event a very low probability. This is further demonstrated by the fact that our precision is slightly worse than the standard approach for $\rho < 0.1$. The main reason here is what we call *particle churn*: *i.e.* as a person sits in the coffee room, some of her particles keep moving out and back into the coffee room, sparking erroneous, low-probability coffee-room events.

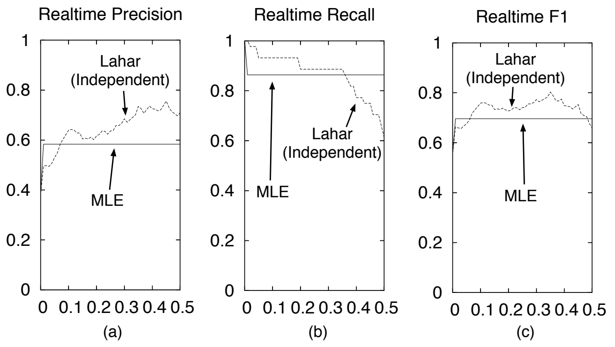


Figure 9: **Precision, recall and F1 as a function of ρ, the threshold. *Real Time* scenario. The baseline is MLE. As the figure shows, our approach improves both recall and precision for values of ρ in $[0.1 : 0.5]$.**

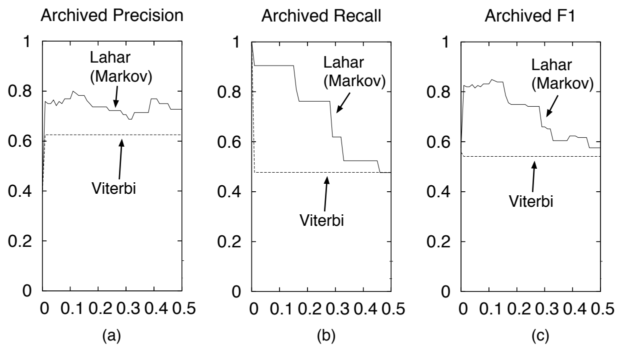
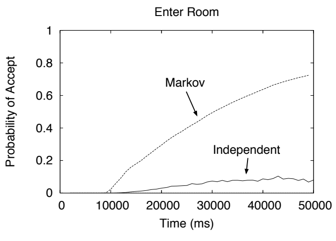


Figure 10: **Precision, recall and F1 for archived streams as a function of ρ, the threshold. The baseline is the MAP obtained using Viterbi. On archived data, the gains of our approach are even more significant.**

Archived Scenario One could reasonably assume that the imprecision in the data is transient and that, given time and processing, we could smooth away the noise and query the resulting deterministic data directly. However, our second series of quality graphs, shown in Fig. 10, demonstrates that even after smoothing, the real data remains imprecise and our approach provides significant gains against Viterbi. For example, at $\rho = 0.12$, we achieve precision gain of 20 points ($\approx$ 33%) and a massive 47-point gain in recall. The gain in precision is due in roughly equal parts to (a) smoothing, and (b) tracking correlations. We verified this by treating the stream independently and noting a resulting 8-point drop in precision. The independent stream maintained comparable recall. The benefit of tracking this imprecision with the richer correlations from smoothing is clearly shown in Fig. 10(c): Our Markovian approach has a higher F1 than Viterbi along the entire interval. Also, the precision and recall are higher than for MAP even for values of



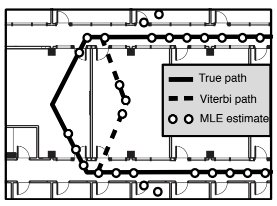


Figure 11: **(a) Acceptance probability at each timestep of the query "Have you been in room 410 for 3 seconds?". Viterbi never accepts the query and so takes value 0 throughout the whole interval. (b) Visual illustration of how MLE and MAP can go wrong. There is no way to decide between rooms, so MAP arbitrarily picks a single path, while MLE hops around due to resampling.**

ρ below 0.1. This is because smoothing has largely eliminated the spurious churning that caused errors in our real-time event detection.

Generalizing to more complicated queries Multiple different queries run on the real data all demonstrated performance numbers similar to those shown above. This is in part because most of the labeled events of interest had characteristics similar to our coffee-room event, *e.g.* when did someone's laptop leave their office, when did a database meeting begin, etc.

A key observation made across multiple queries was that, if a query involved many events which occurred outside of the range of antennas, *e.g.* staying in an office, then correlations became important. To verify this fact we constructed a synthetic trace of a person walking down the hall, entering a particular room, and staying there. We then asked the query: "Were you in the room for k seconds? Because the number of rooms in the vicinity was approximately 6, each room had a probability of approximately 0.15 – however, in the smoothed data, the conditional probability that a person would stay in a room given that she was already there was much higher, *e.g.* 0.6. Thus, the probability that two consecutive timesteps were spent in the rooms was 4 times higher than under independence assumptions, and continued to grow with the number of consecutive timesteps. This explains the trend we see in Fig. 11(a), that the Markovian approach accrues a much higher probability during the visit than independence alone. In contrast, the Viterbi estimate selects the wrong room and misses the event completely.

4.2.2 Discussion

Comparison of MAP and MLE If we compare the absolute recall of MAP and MLE, we see something surprising: the MAP estimate has a lower recall than the MLE[3] This is because the data is very noisy and MAP is forced to pick a path and stick with it. We verified this on synthetic data; a simplified version of this phenomenon is illustrated in Fig. 11. Because the particle mass is essentially split evenly between the two rooms and there are no sensors in the rooms, resampling fluctuations cause the MLE estimate to hop back and forth. In contrast, MAP must pick a single path and so will remain in only one room. An additional factor in the recall disparity is the presence of events that happen for only a few seconds: Events with short duration appear to be spurious, hence are likely to be removed from the MAP path estimate.

Skew The ground truth is data labeled by participants: Each participant in our study carried a laptop and annotated the timestamp

[3]It does not make sense to directly compare the ρ values of the two graphs because calculations of the underlying probabilities differ.

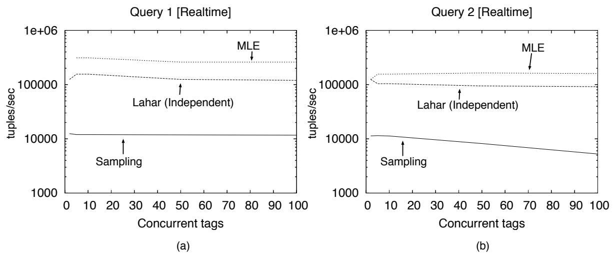


Figure 12: **Comparison of real-time performance techniques. There is less than a factor of two difference between MLE and Lahar. Lahar outperforms random sampling by orders of magnitude.**

at which predefined events occurred, *e.g.* when they entered the coffee room. Ground truth labels are thus noisy. For example, our central query in this section precisely defines an "entered coffee room event" to be triggered the second a person crosses the threshold from the hallway into the coffee room. However in our study, users sometimes first sat down or put away their mugs before reporting that they entered a room. This introduced skew between the labeled times and the real world times. As one would expect, increasing d to tolerate more skew increases the precision and recall of *all* approaches. However, we have verified that the *relative* quality of the different approaches is the same for many different d values.

4.3 Performance

In this section, we evaluate whether the performance of Lahar is suitable for real-time and archived streams. Our main experiment compares Lahar's throughput on synthetic data for two queries: ($Q1$), a Regular Query consisting of a selection on a single stream, and ($Q2$), an Extended Regular Query with a sequence operator. We simulate n objects moving simultaneously for 60 seconds, varying n from 1 to 100. We first measure the raw tuples-per-second throughput achieved by our approach. Our safe plan algorithm requires growing state. It is thus suitable only for stored streams, and we study its performance separately.

Naïve Random Sampling Our default parameters for random sampling are $\delta = 0.1$ (confidence) and $\varepsilon = 0.1$ (precision), *i.e.* with 10% chance per event, we are off by more than 0.1 in our estimation of the underlying probability. In all cases, we run each query seven times and drop the highest and lowest times; the averages reported are of the remaining five runs. The sample variance in all cases is at least two orders of magnitude smaller than the time recorded.

4.3.1 Main Performance Results

Real-Time Streams Fig. 12 shows the comparison of our technique in the real-time scenario, against the MLE and our sampling approach. The graph is in logscale and shows that there is less than a factor of two difference between running with probabilities and running the MLE. This is because the expensive processing operation is the I/O cost of reading the tuples; Lahar's actual term-matching and processing is very efficient. In addition, there is the expected difference between sampling and our technique: exact techniques without expensive randomized draws are several orders of magnitude faster than sampling, even on simple queries. Further, they use drastically lower amounts of memory (see Thm. 3.7). We can conclude from this graph that Lahar provides high-quality event extraction while maintaining viable real-time performance. We see that on the more complicated query $Q2$, the sampling approach already begins to show its unscalable behavior.

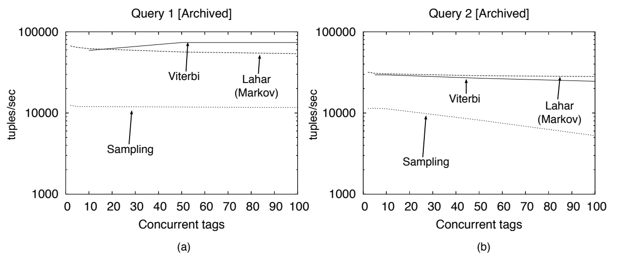


Figure 13: Comparison of archived performance techniques. Lahar performs similarly to Viterbi and is orders of magnitude faster than naïve random sampling.

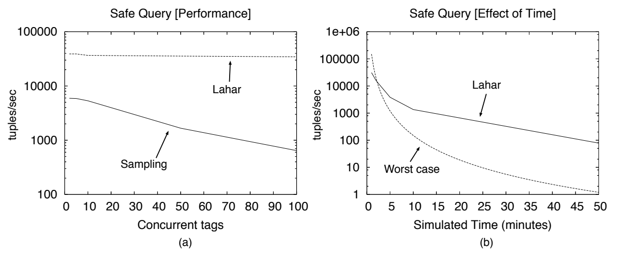


Figure 14: (a) Performance of Safe Query versus sampling (b) Performance of Safe Query as the length of the trace increases.

Archived Streams Fig. 13 shows that the Viterbi algorithm and the Markov algorithm have the same raw throughput, indicating that the overhead of our algorithms is small. Further, the Markovian approach is still orders of magnitude faster than naïve random sampling. However, this comparison is misleading: The Markovian semantic requires many more tuples than the MLE to represent the same number of objects, for two reasons. First, if a timestep contains n tuples, then the CPT stream requires $\approx n^2$ tuples to represent this timestep. Second, the Viterbi algorithm prunes aggressively and keeps only a single tuple per timestep. To capture this we define a measure, the *effective objects per second*, which is computed as the number of people times the number of timesteps, divided by the number of seconds to process the trace. In practice, we found that the effective tuples per second was slower by an order of magnitude (*e.g.* a factor of $9-10$ on these traces). Sampling is affected in exactly the same manner when it uses Markovian correlations.

4.3.2 Additional Experiments and Discussion

Query Complexity Experiment To measure the effect of query complexity, we fixed the number of concurrently-tracked tags to 50 and varied the number of subgoals in an Extended Regular Query. We find that our system can handle queries with up to five subgoals while still keeping pace with the demand of the system. In particular, the time to process the trace is less than the time of the actual trace. We should not expect to scale any better, as we have shown that the query time increases exponentially with the number of subgoals. For Markovian streams, the viable number of subgoals drops to 3 – which is to be expected since Lahar maintain significantly more state. However, since Markovian queries are intended for offline use, this does not pose a significant obstacle.

Safe Plan Experiment Fig. 14(a) shows the performance of a Safe, but not Extended Regular, query $At(p, l_1); At(p; l_2); At(q, l_3)$, on a simulated 60-second trace as we vary the number of concurrent tags. As we can see, even with many objects, the safe plans are much more efficient than the sampling approach. However, since the running time of safe plans depends on the number of timesteps t, we expect that the running time should slow down as the trace length increases. Because each iteration takes time $O(n^2)$, a run of length n should take $\sum_{i=1}^{n} = O(n^3)$. In particular, the throughput should decrease cubically; however, as Fig. 14(b) demonstrates, our asymptotic performance is actually much better than the analytical prediction. This is due to the fact that we evaluate the recurrence lazily.

Optimization Techniques The majority of the bytes of smoothed data (Fig. 8) are actually the result of particle filter delusion during the ten minute break. In the experiments we have presented, Lahar ran directly on the output of the particle filter, including this uninteresting data. However, there are known techniques for reducing storage cost without affecting recall. For example, using a pruning technique [8], we were able to reduce the CPT relation size to $\approx$ 1Gb without a noticeable degradation in quality. We expect that further optimizations along these lines are possible.

5. RELATED WORK

The use of automata as a basis for processing event queries is common to both Cayuga [14, 45] and SASE [46]; however, neither consider probabilistic event streams. Our translation of regular queries is similar to the use of $\exists$ and $\forall$ edges in Cayuga's automaton. However, Cayuga does not identify a class of queries which can be efficiently translated with no state.

MystiQ [9] and Trio [3] are the two most influential probabilistic databases today. MystiQ focuses on query evaluation and so is more closely related to our work. There are two key differences between the projects: (1) In MystiQ the query language is essentially SQL, while our language is essentially the event sequence language of Cayuga, which allows us to query temporal constraints. (2) MystiQ allows events to be either independent or disjoint, but does not allow temporal correlations, such as the Markovian correlations that we consider

Hidden Markov models and their inference algorithms have a long history; we refer the reader to [34] for a survey. Of course, any query we consider here can be encoded as an HMM, but it is unclear how the translation from event queries into HMM inference can be automated. Our focus in this project is *not* to improve inference; instead we focus on managing the results of many different (independent) inference tasks. A project with similar goals is [17], but to the best of our knowledge no algorithmic details are published.

Recent work has started to explore the problem of query processing over probabilistic data streams. Jayram *et al.* [18] introduced the probabilistic stream model. Jayram *et al.* [18, 19] and Cormode and Garofalakis [7] proposed efficient algorithms for computing aggregate functions over uncertain data streams. Their query models are very limited and they do not consider correlations across time. Kanagal and Deshpande [22] support queries over probabilistic streams produced by an HMM but their queries are also limited (selections, projections, and aggregations only). In contrast, our approach supports a much richer query model over probabilistic streams including sequences and joins. We also support correlated streams. Finally, the Data Furnace project [17] outlined a similar goal to ours: extracting probabilistic events from imprecise sensor data. Their design, however, relies on exploiting an inference engine to compute event probabilities while we focus on computing these probabilities efficiently in a query processing engine.

Our name for safe plans follows the papers [9, 10], but our sequencing requires both order and universal quantifiers (*i.e.* no tuple in between), which is not considered by Dalvi and Suciu. There has also been work on probabilistic temporal databases, [13], but the work allows a distribution over the time an event occurred and has a different semantics than the one we present here.

In data mining and stream mining applications [16, 43], the goal is to translate from deterministic but imprecise observations to high-level facts. In our instance, the data itself contains uncertainty with a precise semantics and our goal is simply to answer queries. Further, we study exact algorithms for these problems. It is interesting future work to consider more precise and efficient approximations.

Lahar makes heavy use of particle filters [2] and smoothing techniques [24, 8]. Particle filtering in particular has recently gained attention in databases research, notably [22].

6. CONCLUSION

We have presented Lahar, which enables declarative queries over real time and archived streams of probabilistic events. We demonstrated on real data that our approach provides significant quality benefits versus determinizing the data, while providing efficient execution. The key to our approach was a suite of novel algorithms.

Acknowledgement This work was partially supported by an NSF graduate research fellowship, NSF Grants IIS-0454425, IIS-0513877, IIS-0627585, IIS-0713576, and Gifts from Microsoft including a gift under the SensorMap RFP.

7. REFERENCES

[1] R. Adaikkalavan and S. Chakravarthy. Snoopib: interval-based event specification and detection for active databases. *Data Knowl. Eng.*, 59(1):139–165, 2006.

[2] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[3] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[4] C. Floerkemeier and M. Lampe. Issues with RFID usage in ubiquitous computing applications. In *Proc. of the 2nd Pervasive Conf.*, April 2004.

[5] Cisco Systems. Cisco IOS NetFlow. `http://www.cisco.com/go/netflow`.

[6] Computerworld. Procter & Gamble: Wal-Mart RFID effort effective. `http://www.computerworld.com/action/article.do?command=viewArticleBasic%&articleId=284160`, February 2007.

[7] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proc. of the 2007 SIGMOD Conf.*, June 2007.

[8] R. G. Cowell, S. L. Lauritzen, A. .P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[10] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[11] Dartmouth College. CRAWDAD: A Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.cs.dartmouth.edu/index.php.

[12] S. K. Das et al. The role of prediction algorithms in the MavHome smart home architecture. *IEEE Wireless Communications*, 9(6):77–84, 2002.

[13] A. Dekhtyar, R. Ross, and V. S. Subrahmanian. Probabilistic temporal databases, igr: algebra. *ACM Trans. Database Syst.*, 26(1):41–95, 2001.

[14] A.J. Demers, J. Gehrke, M. Hong, M. Riedewald, and W.M. White. Towards expressive publish/subscribe systems. In *EDBT*, pages 627–644, 2006.

[15] N. Fuhr and T. Rolleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[16] M.N. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE TKDE. Knowl. Data Eng.*, 14(3):530–552, 2002.

[17] Garofalakis et. al. Probabilistic data management for pervasive computing: The Data Furnace project. *IEEE Data Engineering Bulletin*, 29(1), March 2006.

[18] T. S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *Proc. of SODA 2007.*, January 2007.

[19] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *Proc. of the 26nd PODS Conf.*, June 2007.

[20] S. Jeffery et al.. Adaptive cleaning for RFID data streams. In *Proc. of the 32nd VLDB Conf.*, September 2006.

[21] D. Jobst and G. Preissler. Mapping clouds of soa- and business-related events for an enterprise cockpit in a java-based environment. In *Proc. of the 4th Symp. on PPPJ*, pages 230–236, 2006.

[22] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. Technical Report CS-TR-4867, University of Maryland, May 2007.

[23] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *Proc. of Fifth MobiDE Workshop*, June 2006.

[24] M. Klaas, M. Briers, N. de Freitas, A. Doucet, S. Maskell, and D. Lang. Fast particle smoothing: if i had a million particles. In *Proc. of the 23rd ICML*, pages 481–488, New York, NY, USA, 2006. ACM.

[25] M. Lamming and D. Bohm. SPECs: Another approach to human context and activity sensing research, using tiny peer-to-peer wireless computers. In *Ubicomp 2003*, volume 2864, pages 192–199, 2003.

[26] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell Sys. Tech. J.*, 62:1035, 1983.

[27] L. Liao, D. J. Patterson, D. Fox, and H. A. Kautz. Learning and inferring transportation routines. *Artif. Intell*, 171(5-6):311–331, 2007.

[28] J. F. McCarthy and T. D. Anagnost EVENTMANAGER: Support for the Peripheral Awareness of Events. In *HUC*, volume 1927, pages 227–235, 2000.

[29] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.

[30] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[31] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[32] M. Philipose, K. P. Fishkin, M. Perkowitz, D. J. Patterson, D. Fox, H. Kautz, and D. Hahnel. Inferring activities from

interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.

[33] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[34] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1990.

[35] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams (full version). Technical Report 08-03-02, University of Washington, Seattle, WA, 2008.

[36] C. Ré and D. Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *VLDB*, pages 51–62, 2007.

[37] Reuters. Stock Data Feed. `http://about.reuters.com/productinfo/s/stock_data_feed/`.

[38] RFID Journal. Hospital gets ultra-wideband RFID. `http://www.rfidjournal.com/article/view/1088/1/1`, August 2004.

[39] Crossbow Technology. Products: Wireless sensor networks. `http://www.xbow.com/Products/wproductsoverview.aspx`.

[40] University of Washington. RFID Ecosystem. `http://rfid.cs.washington.edu/`.

[41] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, and J.A. Stankovic. An assisted living oriented information system based on a residential wireless sensor network. In *1st Distributed Diagnosis and Home Healthcare (D2H2) Conference*, April 2006.

[42] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, 1967.

[43] F. Wang and P. Liu. Temporal management of RFID data. In *Proc. of the 31st VLDB Conf.*, September 2005.

[44] E. Welbourne, M. Balazinska, G. Borriello, and W. Brunette. Challenges for pervasive RFID-based infrastructures. In *IEEE PERTEC 2007 Workshop*, March 2007.

[45] W. White, M. Riedewald, J. Gehrke, and A. Demers. What is "next" in event processing? In *Proc. of the 26nd PODS Conf.*, pages 263–272, New York, NY, USA, 2007. ACM.

[46] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD*, pages 407–418, New York, NY, USA, 2006. ACM.