On Models and Query Languages for Probabilistic Processes [*]

Daniel Deutch Tova Milo
Tel Aviv University
danielde@post.tau.ac.il milo@post.tau.ac.il

ABSTRACT

Probabilistic processes appear naturally in various contexts, with applications to Business Processes, XML data management and more. Many models for specifying and querying such processes exist in the literature; a main goal of research in this area is to design models that are expressive enough to capture real-life processes and analysis tasks, but at the same time allow for efficient query evaluation. We depict the model established in [13, 16, 17, 18], and claim that it achieves a good balance between expressivity and query evaluation complexity. We compare and contrast the model with other common models for probabilistic processes, highlighting the different choices made in models design and their effect on expressivity and incurred complexity.

1. INTRODUCTION

Probabilistic processes, i.e. processes with some probability distribution over their possible executions, appear naturally in various contexts. For instance, the possible behaviors of web-site users may be captured by such processes, where the probabilities are derived from the popularity of actions among previous users; a recent paper [6] explains how to capture probabilistic XML data via a probabilistic process specification; and there are many additional applications e.g. in areas such as Natural Language Processing [29] or computational biology [35]. Probabilistic processes occurring in such cases typically have intricate structures, and may induce a large (or even infinite, in presence of recursion) number of possible executions [6, 15].

The importance of probabilistic processes along with their complex nature call for the use of a *query language*, that will allow to analyze the processes and their possible executions. The results of such analysis are then applied for debugging the process, optimizing it, or identifying optimal ways to use it. Examples for analysis tasks include *termination* analysis (i.e. finding the probability that the process terminates), identifying likely execution flows (or parts thereof), identifying the probability that the execution reaches a given point, and so on.

To allow for an analysis of the above flavor, three components are required. First, one should design a formal *model* for probabilistic processes and their executions. Second, the model should be accompanied by a corresponding query language that will allow to specify analysis tasks over such processes and executions; the model and query language should be expressive enough to capture real-life processes and analysis tasks, and simple enough to allow for effective formulation of process specification and queries. Third, the model and query language must be supported by efficient query evaluation algorithms. Clearly, there exists a tradeoff between the expressive power of the model and query language, and the complexity of query evaluation. An important goal of research in this area is to find a good balance between the two.

Many models and query languages for probabilistic processes (e.g. [4, 6, 9, 17, 25, 30, 34]) appear in the literature, and vary in their expressive power and the query evaluation complexity they admit. Specifically, in [5, 16, 17] we have suggested a model and query language for *Probabilistic Business Processes (BPs)*, which are used to depict the logical structure of business activities that are common e.g. in e-commerce and in Web Applications [5, 19]. The model is an abstraction of the BPEL [8] standard and allows for an intuitive, graphical representation of the process. The query language is then based on the same graph-based view and allows users to query processes visually, in an intuitive manner, very analogous to how the processes are typically specified. We then studied query evaluation, and provided efficient algorithms, for different fragments of the query language [13, 16, 17].

We claim that our proposed model is successful in

[*]Database Principles Column. Column editor: Leonid Libkin, School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK. E-mail: libkin@inf.ed.ac.uk

achieving a good balance between the expressivity and the complexity of query evaluation. To substantiate this claim, we first informally depict the model and the complexity results achieved for query evaluation. Then, we review other common models and compare their properties. Via this comparison, we highlight the different choices that are made in the models design, and their effects on expressivity and query evaluation complexity.

We next provide a brief overview of the Business Process model and query language.

A Business Process (BP) specification is abstractly modeled as a nested DAG consisting of activities (nodes), and links (edges) between them, that detail the execution order of the activities [15, 23]. For example, consider a BP of an on-line travel agency. The BP specification may include activities (nodes) for flight and hotel reservation, car rental, payment and confirmation services, and edges that describe their execution order. The DAG shape allows to describe parallel computations. For instance, advertisements may be injected in parallel to the search. BP activities may be either atomic, or compound. In the latter case their possible internal structures, called *implementations*, are also detailed as DAGs, leading to the nested DAG structure. A compound activity may have different possible implementations, corresponding to different user choices, variable values, servers availability, etc. An *Execution Flow* (abbr. EX-flow) is then an actual running instance of a BP, obtained by choosing a single implementation for each compound activity. A BP specification induces a set of such possible EX-flows; this set may be large, or even infinite when the BP specification contains *recursion*.

In practice, some EX-flows are more common than others. This is modeled by a probability distribution over the possible implementations of compound activities [16, 17]. A BP specification along with a description of such distribution is called a *Probabilistic BP*. We note that the probabilities of choices dictating the execution course are, in typical cases, *inter-dependent*; for instance a user making a reservation for a flight to Paris is more likely to also reserve an hotel in Paris. We allow to model such dependencies in a straightforward manner.

We also defined a query language for analyzing BPs. Queries are used to define EX-flows or parts of them, that are of interest to the analyst. For our travel agency example, an analyst may wish to find out *how is one likely to book a travel package containing a flight reservation?*, or *how is this likely to be done for travelers of a particular airline company, say British Airways?*. There may be many different ways for users to book such travel packages. But assume, for instance, that we obtain that a likely scenario for British Airways reservations is one where users first search for a package containing both flights and hotels, but eventually do not reserve an hotel. Such a result may imply that the combined deals suggested for British Airways fliers are unappealing, (as users are specifically interested in such deals, but refuse those presented to them), and can be used to improve the Web site. Queries are defined using *execution patterns* (abbr. EX-patterns), generalizing EX-flows similarly to the way tree patterns, used in query languages for XML, generalize XML trees [10]. In more details, EX-patterns bear the structure of an EX-flow, where activity names are either specified, or left open using a special ANY symbol and then may match any node. Edges in a pattern are either regular, interpreted over edges, or transitive, interpreted over paths. Similarly, activities may be regular or *transitive*, for searching only in their direct internal flow or for searching in any nesting depth, respectively. A match of the query is captured via the notion of an *embedding*, which is a homomorphism from an EX-pattern to an EX-flow, respecting node labels and edge relation. We then note that given a BP specification s and a query q, the number of possible EX-flows of s that qualify according to q may be extensively large, or even infinite when s is recursive. In practice, analysts are only interested in the possible EX-flows of s that are most likely to occur in practice. This is in fact a flavor of *top-k analysis*.

In many cases, analysts are further interested only in some *part* of the possible EX-flows. For that, we define top-k *projection* queries, whose output consists of the likely sub-flows. An important question that rises when considering projection queries is the choice of a ranking metric for the query results. Recall that our model associates a likelihood value with every possible EX-flow of the BP specification; with projection queries, a single projection results may originate from multiple EX-flows. In this case, the likelihoods of all such EX-flows should then be *aggregated*, to form the result score. Evidently, different choices of such aggregation functions require different query evaluation mechanisms and incur different complexity of query evaluation. We consider the *max* and *sum* aggregation functions, explain their intuitive meaning and depict the complexity of query evaluation for each choice of function.

Paper Organization. In Section 2 we recall our model of probabilistic BPs and queries over such BPs; in Section 3 we overview the complexity of query evaluation for such queries. In Section 4 we overview other common models for probabilistic models, comparing them to our BP model. We conclude in Section 5.

2. MODEL

We (informally) depict in this section the model of probabilistic Business Processes (BPs), and a query language over such processes, originally defined and used in [5, 16, 17, 18]. Additional process models and their connection to this model are discussed in Section 4.

2.1 Business Processes

We start by depicting the model for Business Processes (without probabilities), then introduce probabilities. At a high-level, a BP specification encodes a set of activities and the order in which they may occur. A BP specification is modeled as a set of node-labeled DAGs. Each DAG has a unique start node with no incoming edges and a unique end node with no outgoing edges. Nodes are labeled by activity names and directed edges impose ordering constraints on the activities. Activities that are not linked via a directed path are assumed to occur in parallel. The DAGs are linked through implementation relationships; the idea is that an activity a in one DAG is realized via the activities in another DAG. We call such an activity *compound* to differentiate it from *atomic* activities which have no implementations. Compound activities may have multiple possible implementations, and the choice of implementation is controlled by a a condition referred to as a *guarding formula.*

Example 2.1. *Fig. 1 depicts a partial BP specification. Its root DAG consists of a single activity* `chooseTravel`. `chooseTravel` *is a compound activity having 3 possible implementations F_2, F_3, F_4. These correspond to different choices of travel search (flights only, flights + hotels, or flights + hotels + cars) and are guarded by corresponding formulas. The idea is that exactly one formula is satisfied at run-time (the user chooses one of the three search types) and thus* `chooseTravel` *is implemented either by F_2, F_3 or F_4. Consider for example F_1; it describes a group of activities comprising user login, the injection of an advertisement, the* `Flights` *activity, and the* `Confirm` *activity. Directed edges specify the order of activities occurrence, e.g. users must login before choosing a flight. Some of the activities (e.g.* `Advertise` *and* `Flights`*) are not in a particular order, and thus may occur* in parallel. `Login` *and* `Advertise` *are atomic whereas* `Flights` *and* `Confirm` *are compound. Note that the specification is recursive as e.g. F_2 may call F_1.*

We note that satisfaction of guarding formulas is determined by external factors, such as user choices. We assume that exactly one guarding formula can be satisfied when determining the implementation of a given compound activity occurrence, but satisfaction of guarding formulas can change if activities occur several times. For instance, a user may choose to search for flights and hotels the first time she goes through F_1 and for flights only the second time.

Execution Flows. An Execution Flow (EX-flow) is modeled as a nested DAG that represents the execution of activities from a BP. Since, in real-life, activities are not instantaneous, we model each occurrence of an activity a by two a-labeled nodes, the first standing for the activity *activation* and the second for its *completion* point. These two nodes are connected by an edge. The edges in the DAG represent the ordering among activities activation/completion and the implementation relationships. To emphasize the nested nature of executions, the implementation of each compound activity appears in-between its activation and completion nodes. An EX-flow structure must adhere to the structure of the BP specification, i.e., activities must occur in the same order and implementation relationships must conform to τ.

Example 2.2. *Two EX-flows of the travel agency BP are given in Fig. 2. Ordering edges (implementation edges) are drawn by regular (resp. dashed) arrows. Each EX-flow describes a sequence of activities that occurred during the BP execution. In Fig. 2(a) the user chooses a "flights+hotels" search, reserving a "British Airways" flight and a "Marriott" hotel, then confirms. Fig. 2 (b) depicts another possible EX-flow, where the user chooses a "flights only" search, followed by a choice of British Airways flight, but then resets and makes other choices (omitted from the figure).*

Likelihood. We note that the EX-flow occurring at run-time is dictated by the truth values of guarding formulas, which in turn dictate the choice of implementation for compound activities (out of the possibly multiple implementations associated with each such activity in the BP specification). In a probabilistic process, each such guarding formula is associated with a probability of being satisfied at run-time, and this probability may be dependant

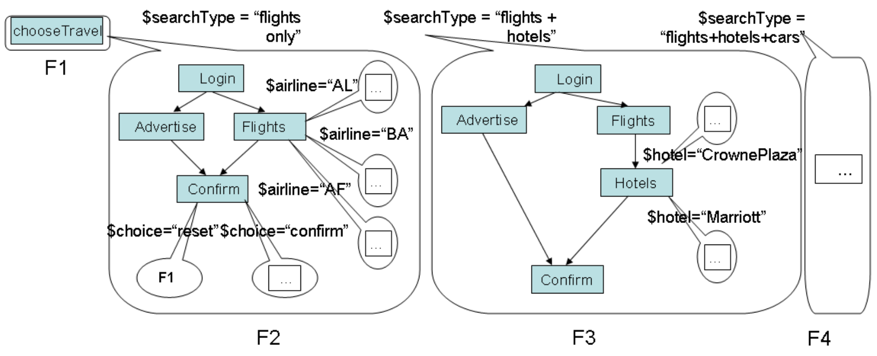

Figure 1: Business Process Specification

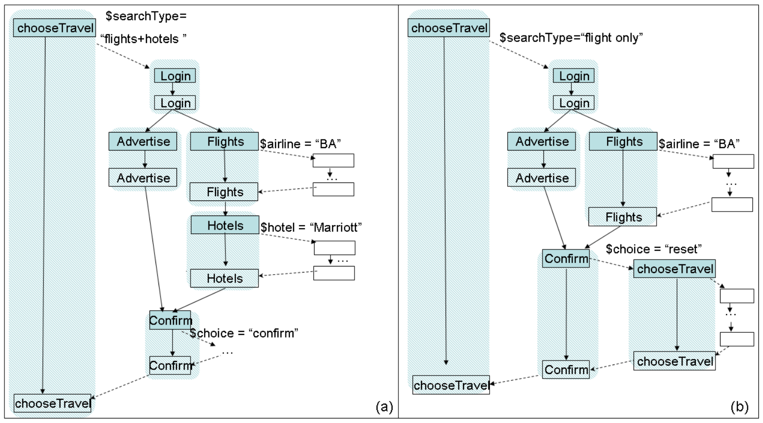

Figure 2: Execution Flows

on the EX-flow thus far and formulas previously satisfied.

To model this we use two likelihood functions. The first, named c–likelihood (choice likelihood), associates a probability value with each guarding formula (implementation choice), describing the probability that the formula holds, *given the partial EX-flow that preceded it*. The second, named f–likelihood (flow likelihood) reflects the joint likelihood of satisfaction of guarding formulas occurring along the flow and is defined as the multiplication of c–likelihood values. A BP specification along with a likelihood function over the guarding formulas occurring in the specification is called a *Probabilistic Business Process*.

Note that the c–likelihood function δ receives two inputs: a guarding formula f in hand, and the partial EX-flow e' that preceded the implementation choices guarded by f. This allows us to capture *dependencies* among likelihood values. We distinguish between three classes of c–likelihood functions, according to their sensitivity to e' (referred to as "history").

History independence: History-independent functions imply probabilistic independence between the choices of implementation in different activities occurrences throughout the flow. More formally, a c–likelihood function is history-independent for every guarding formula f, c–likelihood (e,f) = c–likelihood (e',f) for each two EX-flows e, e'.

Bounded-history: Bounded-history functions capture the more common scenario where the c–likelihood value does depend on the history e of the EX-flow, but only in a bounded manner. That is, to determine the c–likelihood of any implementation choice at an activity node n, it suffices to consider only the implementation choices of the last b preceding compound activities, for some bound b. By "preceding" we refer here to activity nodes $\hat{n}$ that are *ancestors* of n in e (in contrast to nodes that occur in parallel and thus in general may or may not precede n). By "choice" we refer to the formula guarding the implementation selected for the activity.

Unbounded-history: Unbounded-history functions may use an *unbounded* portion of the flow history

e to compute the next choice's likelihood. For instance, if the price of a given product depends on the *exact full sequence of searches that the user performed prior to the purchase*, then the corresponding c–likelihood function is unbounded-history.

Observation. Note that, by definition, for non-recursive BPs, c–likelihood functions are always bounded-history, with the bound being, at most, the BP nesting depth. Recursive BPs, on the other hand, may have unbounded-history c–likelihood functions. In practice, typical c–likelihood functions are bounded-history, and moreover the bound is typically small [36].

2.2 Query Language

We next consider a query language for Probabilistic Business Processes, originally defined in [5] and further refined in [17, 18]. We start by considering *selection queries*, whose output is the (representation of the) set of all EX-flows of the original BP specification, that also match the query. We then introduce *projection queries*, that further allow to focus on some sub-flows that are of interest.

Queries are defined using *execution patterns*, an adaptation of the tree-patterns of existing XML query languages, to nested EX-flow DAGs. Such patterns are similar in structure to EX-flows, but contain *transitive edges* that match any EX-flow *path*, and *transitive activity nodes*, for searching deep within the implementation subgraph, of the corresponding compound activities, at any level of nesting. Nodes/ formulas in the pattern may be labeled by the wildcard ANY and then may match any EX-flow node/formula.

EXAMPLE 2.3. *The query in Fig. 3 (a) (ignore for now the rectangle surrounding a sub-graph of the pattern) describes EX-flows that contain a choice of some British Airways (abbr. "BA") flight followed by a confirmation. The double-lined edges are* transitive edges*, and may match any sequence of activities. The doubly bounded* `chooseTravel` *activity is a* transitive activity*. Its implementation nodes may be embedded anywhere inside the implementation of the corresponding EX-flow activity, at any nesting level.*

The matching of an EX-pattern p to an EX-flow e is called an *embedding*. An embedding of p into e is a homomorphism ψ from nodes and edges in p to nodes,edges and paths in e such that the root of p is mapped to the root of e; activity pairs in p are mapped to activity pairs in e, preserving node labels and formulas; a node labeled by ANY may be mapped to nodes with any activity name. For non-transitive compound activity pairs in p, nodes in their direct implementation are mapped to nodes in the direct implementation of the corresponding activity pair in e. As for edges, each (transitive) edge from node m to n in p is mapped to an edge (path) from $\psi(m)$ to $\psi(n)$ in e.

We then distinguish between selection and projection queries, as follows.

Selection Queries. A selection query is represented by an EX-pattern. An EX-flow e belongs to the query result if there exists some embedding of p into e. We then say that e *satisfies* p. When evaluated against a Business Process s, the output of p consists of all full EX-flows of s that also satisfy p; the set of top-k most likely out of these EX-flows are denoted top-k(s,p).

Projection Queries. In many cases, analysts are not interested in full execution flows of the process, but rather in focusing on some part of them. To that end, we define projection queries over probabilistic processes. For selection queries, the query result consisted of the full EX-flow in which the EX-pattern was embedded. We generalize the definition of such queries and allow a specific part of the pattern to be specified as the *projected part.* The rest of the pattern serves for selecting EX-flows of interest. Namely, only EX-flows in which an embedding of the entire pattern are considered; within these flows, only nodes and edges matching the projected part are in fact projected out and appear in the query result.

Given an embedding ψ of a query q in an EX-flow e, the embedding result is then defined as the nodes and edges of e to which ψ maps the nodes and edges of the projected part of q. For an EX-flow e, the result of q on e, denoted $q_{\downarrow}(e)$, consists of the results of all possible embeddings of q into e; finally, for a BP s, the result of q on s, denoted $q_{\downarrow}(s)$ is the union of all possible results of q when applied on the EX-flows of s. Namely $q_{\downarrow}(s) = \bigcup_{e \in flows(s)} q_{\downarrow}(e)$ where flows(s) is the set of possible EX-flows of s.

Note that an EX-flow $e' \in q_{\downarrow}(s)$ may originate from several EX-flows of s, namely there may be several $e \in flows(s)$ s.t. $e' \in q_{\downarrow}(e)$. Before defining the top-k projection results, we should first decide on how to aggregate the weights of these individual EX-flows, to form the score of projection result. We consider two possible aggregation functions (and consequently, semantics of queries), *max* and *sum.* Under *max* semantics, the *score* of e' is defined as $score(e') = max\{f\text{-}likelihood(e) \mid e \in flows(s) \wedge e' \in q_{\downarrow}(e)\}$. Under *sum* semantics,

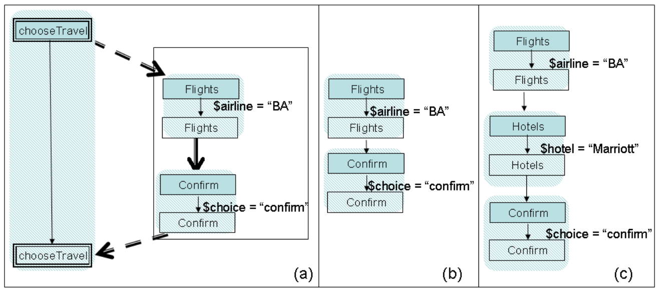

Figure 3: Query

$score(e') = \sum_{e \in flows(s) \wedge e' \in q_{\downarrow}(e)} f\text{-}likelihood(e)$.

The top-k results of a projection query q over a BP specification s, with respect to max (sum) semantics, are then denoted top-k$_{max}(q_{\downarrow}, s)$ (top-k$_{sum}(q_{\downarrow}, s)$). When the semantics used is clear from context, we omit it from notation and simply write top-k$(q_{\downarrow}, s)$.

Example 2.4. *Let us consider again the EX-pattern in Figure 3(a), this time as a query, with the rectangle denoting its projected part. Note that the projection focuses on the execution sub-flows that may occur between the point where a user chooses a "BritishAirways" flight and the final confirmation of her reservation. Note that, due to recursive nature of the BP, there are in general an infinite number of such possible sub-flows (query answers) - a user may reset and restart the search an unbounded number of times. Two possible answers to the query appear in Fig. 3 (b) and (c). The first answer corresponds to users that choose at some point a "flightsOnly" search, pick a "BA" flight and then immediately confirm. The second answer corresponds to users that choose at some point a "flights+hotels" search, pick "BA" as airline and Marriott as hotel, and confirm.*

Observe that each of these answers (sub-flow) have potentially an infinite number of possible origin EX-flows, differing in what the user had done prior to/after the reservation. The likelihood of each answer is the max / sum of likelihoods of all these origins, according to the semantics in hand.

To illustrate the difference between the two semantics (max and sum), consider the above query and a case where one particular deal *D, consisting of a specific flight, hotel and car rental reservation, is very popular, but where packages consisting of only flight and hotel reservations (with no car rental) are* overall *more common (even though each given offer is individually less popular than the specific flight+hotel+car deal D). With* sum *semantics, the result corresponding to the flight+hotel option is ranked highest, as it appears in most EX-flows. But with* max *semantics, flight+hotel+car will be ranked highest, as there exists one very popular EX-flow in which it appears.*

3. QUERY EVALUATION

We next review the main results on query evaluation over probabilistic Business Processes. We omit the proofs and algorithms and refer the reader to prior publications for the full details.

There are three axes that determine the complexity of query evaluation: whether the BP specification is recursive or not, the class of the c–likelihood function used in the process description (i.e. its level of dependency over the history), and the query semantics (selection vs. projection). Interestingly, when considering projection queries, the choice of aggregation function also bears a significant effect on the complexity of query evaluation. The complexity results are summarized in Table 1.

We start the discussion with selection queries, then proceed to projection queries.

3.1 Selection Queries

Given a BP specification s, a selection query q, and a number k, the top-k results of q with respect to s (denoted $top\!-\!k(q, s)$) are defined as the k most likely flows of s, out of those in which an embedding of q exists. [1]. We refer to the problem of identifying top-k(q,s) (given the above input) as `TOP-K-ANSWERS`.

We next discuss the problem complexity for the different classes of c–likelihood functions.

History-independent c–likelihood function. When the c–likelihood function is history-independent, we can design an efficient algorithm for top-k selection query evaluation, as the following theorem

[1]Certain EX-flows may have equal weights, which implies that there may be several valid solutions to the problem, in which case we pick one arbitrarily.

BP	Weight Function	Queries	Data Complexity	Query Complexity
(Non-)Recursive	History-independent	Selection / Projection (max)	PTIME [16, 17, 18]	EXPTIME, NP-hard [14]
(Non-)Recursive	Bounded-History	Selection / Projection (max)	PTIME in BP, EXPTIME (NP-hard) in bound [16, 17]	EXPTIME
Recursive	Unbounded-History	Selection / Projection	Undecidable [16, 17]	Undecidable
(Non-)Recursive	History-independent	Projection (sum)	EXPTIME, NP-hard [13]	EXPTIME
Non-Recursive	History-independent	Projection (sum) restricted	PTIME (unit-cost rat. arithmetic) [13]	EXPTIME
(Non-)Recursive	Bounded-History	Projection (sum)	EXPTIME in BP, 2-EXPTIME in bound [13]	EXPTIME

Table 1: Complexity of query evaluation for Business Processes

holds (the algorithm proving the theorem correctness originally appeared in [16] and was improved in [18]).

THEOREM 3.1. *[16, 18] Given a probabilistic BP s with a history-independent c–likelihood function, and a query q, we may solve* `TOP-K-ANSWERS` *in time polynomial in the size of s and in the output size, and exponential in the query size.*

The algorithm solving `TOP-K-ANSWERS` works by "intersecting" the BP specification s with the query q, outputting a BP specification s' whose EX-flows are exactly the EX-flows of s in which there exists an embedding of q. The algorithm then employs a sophisticated A*-style analysis that explores the space of possible EX-flows of s', by repeatedly testing possible expansions of activities and avoiding infinite loops by maintaining a table of the previously computed sub-flows in a dynamic programming style.

We can also show that a PTIME algorithm w.r.t. query size is not possible, unless P=NP. For that, we define the corresponding decision problem `BEST-ANSWER`, which tests, whether the top-1 EX-flow of a given BP specification is more likely than some threshold t. The following theorem holds (the proof appears in [17], following a construction from [14], using reduction from 3-SAT):

THEOREM 3.2. *[14, 17]* `BEST-ANSWER` *is NP-hard when the input size is considered to be the query size (and the BP specification size is considered a constant).*

Bounded-History c–likelihood function. Bounded-History c–likelihood functions pose further challenges, as the c–likelihood of each choice may depend on a number of other choices. The following theorem was shown in [17]:

THEOREM 3.3. *[17] Given a BP s, a bounded-history c–likelihood function δ with bound b, and a query q, we may solve* `TOP-K-ANSWERS` *in time polynomial in the size of s and in the output size, and exponential in b and in the query size.*

The general idea of the algorithm is to create, a new BP s', with a new, *history-independent*, c–likelihood function δ', such that s and s' have essentially the same set of flows with the same f–likelihood . Then, we apply the algorithm from the proof of Theorem 3.1. To obtain s' and δ' we annotate the activity names in the specification, "factoring" within the names all information required for the computation of c–likelihood of formulas, namely a pre-condition vector, including the m last choices for all activities. Additionally, the new activities names also contain post-condition vectors, necessary to assure consistencies between pre-conditions assumed by activities and what has happened in their predecessors.

It was further shown in [17] that the added exponential dependency on the history-bound b is inevitable, as the following theorem holds (proof by reduction from Set Cover):

THEOREM 3.4. *[17] For bounded-history c–likelihood functions with bound b,* `BEST-ANSWER` *is NP-hard when the number of activities in the BP specification and in the query are considred to be constants, and b is considered to be the input size.*

Unbounded-History c–likelihood functions. Last, for unbounded-history c–likelihood functions, we showed in [16] that `TOP-K-ANSWERS` becomes impossible to solve, as the following theorem holds:

THEOREM 3.5. *[16] If the c–likelihood function given as input may be unbounded-history, then* `BEST-ANSWER` *is undecidable.*

The proof is by reduction from the halting problem, where we encode a Turing Machine as a BP specification with unbounded-history c–likelihood function; the latter is used to model the TM tape.

3.2 Projection Queries

We next turn to projection queries. It is easy to observe that all hardness results presented above for selection queries, also hold for projection queries. However, it turns out that the upper bounds depend upon the aggregation function in use. Specifically, for the *max* semantics:

THEOREM 3.6. *[17] Theorems 3.1 and 3.3 hold for projection queries with* max *semantics.*

To prove Theorem 3.6 we adapt the algorithms that were designed for selection queries to account for projection queries with max semantics. A naive attempt for such adaptation involves selection followed by projection, namely the generation of a BP specification whose EX-flows are the top-k selected EX-flows, then projecting these EX-flows to find the top-k projections. However the size of the selected EX-flows, and consequently the algorithm complexity is exponential in the size of the input BP specification, even when the projection results themselves are of small size. Moreover, we may show that it is infeasible to compute a BP specification whose EX-flows correspond to *all* projection results and then retrieve the top-k results out of them, intuitively because projection over transitive edges may result in exponentially many paths which cannot be compactly represented together. However, we may still perform a two-steps algorithm, similar to the one employed for selection queries: he first step of the refined evaluation algorithm generates a specification that captures only a *subset* of the projection results, where for each transitive edges only the top-k matching paths are kept; this subset includes in particular the top-k projections. We can then find the top-k EX-flows of this specification, which are the top-k projection results with respect to max semantics.

When the *sum* aggregation function is used, query evaluation becomes much more difficult. Specifically, we can show the following:

THEOREM 3.7. *[13] For projection queries with* sum *semantics,* `BEST-ANSWER` *is NP-hard (under Turing computation model) in the BP size, even for non-recursive specifications and for queries with no transitive nodes.*

The proof is by reduction from 3-SAT. Interestingly, the problem is hard even for non-recursive BPs. This is because the hardness is due to the DAG structure of the BP graphs, that may lead to exponentially many paths being projected over the query transitive edges; consequently the sum of likelihoods of exponentially many EX-flows must be computed. See the comparison in Section 5 to Probabilistic XML.

On the other hand, under certain plausible assumptions over the BP specification structure, we can design an algorithm whose complexity is EXPTIME in the BP specification size. We say that a projection query q w.r.t. a BP specification s has *separated likelihoods*, if for every two query answers, either they have equal likelihoods or the difference of their likelihoods is greater than some fixed constant ϵ. The following theorem then holds:

THEOREM 3.8. *[13] For projection queries with sum semantics and BP specifications with history-independent c–likelihood function:,* `TOP-K-ANSWERS` *may be solved in EXPTIME, for: (1) non-recursive BP specifications, and (2) recursive BP specifications, when the query has separated likelihoods w.r.t. the specification.*

The algorithm uses a small world theorem, showing that among the infinitely many paths that may match the query transitive edges, it is sufficient to examine paths whose length is bounded by the size of the BP specification multiplied by k (the number of required results); consequently only exponentially (in the above quantity) many paths are examined. Then, the algorithm reduces `TOP-K-ANSWERS` to the computation of likelihoods for a set of exponentially many candidate answers that are represented as boolean queries, i.e. queries for which we ask only for the likelihood of a match; finally, techniques from [25] are adapted to approximate the likelihood values of these boolean queries up to a point allowing to separate the different candidate answers (this is why separated likelihoods are required). Exact computation of likelihood values for boolean queries is impossible here since they may in general be irrational [13], thus we must resort to their approximation.

Note that the proof for NP-hardness (Theorem 3.7) used queries with transitive edges. When no transitive edges are allowed in the query, our algorithm is more efficient, for non-recursive BP specifications. Specifically, consider the unit-cost rational arithmetic model [7]: in this model, when computing the complexity, we assume that each arithmetic operation on rational numbers may be done in O(1), regardless of the numbers size. The following theorem then holds:

THEOREM 3.9. *[13] When the query projected part does not include transitive edges, and the BP specification is non-recursive,* `TOP-K-ANSWERS` *for projection queries with* sum *semantics may be solved* *in PTIME under the unit-cost rational arithmetic model.*

The assumption of unit-cost rational arithmetic is necessary here, as there are examples where even non-recursive BP specifications and queries without transitive edges yield projection results with probabilities whose encoding is exponential in the BP specification size.

When the c–likelihood is bounded-history, we may first use the construction depicted above for proving Theorem 3.3 to obtain a new BP specification s' with history-independent c–likelihood function, then apply the above algorithm. Note that since the size of s' may be exponential in the history-bound, and the algorithm that finds the top-k projections may incur exponential time in the size of its input, the overall complexity is double-exponential in the size of the bound. The following theorem holds:

THEOREM 3.10. *[13] For projection queries with sum semantics and BP specifications with bounded-history c–likelihood function with bound b,* `TOP-K-ANSWERS` *may be solved in time exponential in the size of the BP specification and double exponential in b, for: (1) non-recursive BP specifications, and (2) recursive BP specifications, when the query has separated likelihoods w.r.t. the specification.*

4. ADDITIONAL MODELS

We have depicted above our model for probabilistic Business Processes. The literature contains a variety of formalisms for *probabilistic process specifications*. We next discuss a representative subset of these models; for each model we review the query languages used for its analysis, and the corresponding complexity results for query evaluation. A summary of the models and results reviewed in this Section appears in Table 2. We also compare and contrast these models with the model of probabilistic BP.

4.1 Markov Chains

The probabilistic counterpart of Finite State Machines (FSMs) is called Markov Chains (MCs) [30]. MCs are in fact FSMs, but with transitions associated with probabilities; these state machines bear the Markovian property [30], meaning that the choice of transition is independent of the previous states that were traversed, given the current state. The common approach in analysis of such processes is to use a query language based on *temporal logic* [33], e.g. LTL, CTL^* or μ-calculus (these differ from each other in terms of expressive power, see e.g. [33]). Query evaluation for such logics then correspond to computing the probability that the property defined by the given query, holds for a random walk over the Markov Chain. It is known that query evaluation of such temporal logic queries over a Markov Chain may be performed in time polynomial in the size of the Markov Chain, and exponential in the query size [11].

Comparison to the BP model and query language. Finite State Machines (and consequently, Markov Chains) lack the expressive power to capture the possibly recursive processes allowed in the BP model here (we shell see in the sequel that BPs are equivalent to "stronger" models). The definition of history-unbounded c–likelihood functions resembles (and is inspired by) the Markovian Property.

As for the query language, a first observation is that temporal logic allows only for *boolean queries* (i.e. computing the probability that a given boolean property holds), while selection and projection queries of the kind depicted in Section 2 were not studied in the context of Markov Chains (to the best of our knowledge). But even when considering just boolean queries, there are queries expressible in our language but not in temporal logic. To that end, note that temporal logics are *bisimulation-invariant*, intuitively meaning that they can query only the process "behavior" rather than its structure (see [14] for exact notions). In contrast, our query language takes a database approach and allows to further query the structure of the process as well as the structure of its possible executions. Indeed, we have shown in [14] that some queries expressible in our query language are not expressible in temporal logic (and vice versa).

4.2 Introducing Function Calls

As stated above, Markov Chains are the probabilistic counterpart of Finite State Machines. Similarly, there exist probabilistic counterparts to stronger model; these models allow for function calls, and in their full-fledged version, also for recursion. Specifically, *Recursive Markov Chains* (RMCs) [25] extend Markov Chains to allow for recursive invocations of procedures. An RMC consists of a collection of finite state components, each of which is a Markov chain, that can call each other in a potentially recursive manner. The authors of [25] further show that RMCs generalize (and can express) previous important models for probabilistic processes such as probabilistic Pushdown Automata [9], and Stochastic Context Free Grammars [32].

Model	Query	Data Complexity	Query Complexity
MC	Temporal	PTIME	EXPTIME
RMC	MSO	EXPTIME, at least as hard as SQRT-SUM	Non-elementary
HMC	MSO	PTIME (with unit-cost relational arithmetic)	Non-elementary
Tree-like HMC	MSO	PTIME	Non-elementary
Prob. XML [31, 37] (non-recursive)	Selection / Projection	PTIME	$\sharp P$-complete

Table 2: Complexity of query evaluation for different models

Query evaluation over RMCs was first studied in [25] for approximating the probabilities of reachability and termination. In a recent work, [6] has extended the analysis for a very strong query language, namely the Monadic Second Order (MSO) Logic. Denote as MSO-EVAL the problem of computing the probability that a given MSO query is satisfied in a random execution of an RMC; the following theorem then holds:

THEOREM 4.1. *[24] MSO-EVAL can be performed in EXPTIME Data Complexity and non-elementary query complexity.*

It is highly unlikely that query evaluation for RMCs may be solved in PTIME, as [25] showed that this problem is at least as hard as `SQRT-SUM`. `SQRT-SUM` is the problem of deciding, given natural numbers $(d_1, ..., d_n)$ and a natural number k, whether $\sum_{i=1,...,n} \sqrt{d_i} \leq k$, strongly believed not be solvable in PTIME under Turing Computation Model [27]. Interestingly, this hardness result holds even for very simple reachability and termination analysis, and even when we are only interested in approximating the probabilities.

Also note in this context the non-elementary query complexity of the evaluation problem. This complexity is entailed by the use of the highly expressive MSO logic as a query language, and is unavoidable even for weaker process models due to [38].

Several restricted variants of RMCs were studied in [6, 25]; specifically, HMCs (Hierarchical Markov Chains) is a restricted version that does not allow for recursive calls. Interestingly, query evaluation here is still harder than for non-hierarchical MCs. This is due to the fact that the probability of a given property may be so small that exponentially many bits are required to represent it. This, in turn is due to the function-like structure of HMCs that allow several functions to call a single function; in this way, exponentially many call paths may be represented compactly.

Thus, for the Turing computational model there is no hope for a PTIME data complexity algorithm. However, if we use the unit-cost rational arithmetic model [7], we do not have to care about the size of numbers. The following theorem then holds:

THEOREM 4.2. *[6] MSO-EVAL for HMC can be performed in PTIME Data Complexity (and non-elementary query complexity), under the unit-cost rational arithmetic model.*

If the HMC possible calls graph has a tree-shape, then the above problem is avoided and we obtain a PTIME data complexity under the conventional Turing Computational Model

THEOREM 4.3. *[6] MSO-EVAL for Tree-Like HMCs ([6]) can be performed in PTIME Data Complexity (and non-elementary query complexity), under the Turing Computational Model*

Comparison to the BP model and query language. Our Business Process model with history-independent likelihood functions may be shown to be equivalent to a restricted version of RMCs, namely 1-exit RMCs (the proof is an adaptation of Theorem 3 in [14] to a probabilistic context), and similarly non-recursive Business Processes are equivalent to 1-exit HMCs. To our knowledge, no extended model for RMCs that allows for dependencies in-between probabilistic choices was studied. As for the query language, our language (when queries are considered as boolean) is contained in MSO in terms of expressive power. But note that the use of MSO is very costly: it incurs non-elementary query complexity, with respect to the EXPTIME obtained for our language. Our query language is restricted, but is expressive enough to express interesting and practical properties as shown in [5], while allowing for a practically efficient evaluation. Furthermore, as the case for MCs, selection and projection queries were not studied in the context of RMCs (to our knowledge).

4.3 Probabilistic XML

It turns out that there are intricate connections between Probabilistic XML and probabilistic processes. Specifically, the recent work of [6] mentioned above establishes a model for Probabilistic XML that is based on Recursive State Machines. Then, in addition to their study of MSO queries, they study more restricted languages which are common for XML querying such as (boolean) Tree Pattern Queries and XPath, and show that they incur lower query complexity: $FP^{\sharp P}$ (the class of computation problems that can be solved in polynomial time using a $\sharp P$ oracle) for tree pattern queries and PSPACE for XPath.

Previous works on Probabilistic XML documents [2, 3, 31, 37] used non-recursive models for expressing distributions over a finite collection of possible documents. In [31], a p-document is an XML tree with two types of nodes: ordinary nodes which are just regular XML nodes, and distributional nodes that define some probabilistic distribution over subsets of their children. A random XML document may then be generated by making probabilistic choices that follow these distributions. The authors of [31] then study query evaluation over p-document and use a language that allows for selection and projection; the semantics used there for projection is the counterpart of our *sum* semantics, and they show that query evaluation is feasible, as follows:

THEOREM 4.4. *[31] Query Evaluation over p-documents may be done in time complexity that is polynomial in the p-document size, and exponential in the query size.*

They also show that the exponential dependency on the query size is inevitable, unless $P = \sharp P$.

Comparison to the BP model and query language. An important distinction between the model of [31] and our BP model is that BP graphs are DAG-shaped while XML graphs are tree-shaped. Evidently, this difference causes an unavoidable (unless P=NP) exponential overhead in query evaluation, and also complicates the algorithmic construction (see [13] and the discussion in Section 3). While [6] considers XML with sharing, a model that does allow for DAG-shape graphs in the specification, the weaker query languages (tree patterns and XPath) studied in this cannot capture the DAG-shaped patterns expressible in our query language; also, [6] only studies boolean queries in contrast to the selection and projection queries allowed in our language.

5. CONCLUSION

We have reviewed in this paper several important models for specifying and querying probabilistic processes and compared them in terms of expressive power and complexity of query evaluation. We further depicted our proposed model for Business Process and our query language that allows for top-k selection and projection queries. We then explained how our proposed model achieves a reasonable balance between expressivity and evaluation complexity.

Clearly, more work has to be done to capture real-life probabilistic processes and analysis tasks. In particular, while the models consider, to some extent, the data manipulated in the process execution, more work is required for modeling and querying such manipulated data in real-life settings. The modeling (and analysis) of the data manipulated by processes was studied in e.g. [12, 20, 21, 22, 26, 28], but the processes studied there were not probabilistic; combining these two lines of research into a unified framework for probabilistic processes along with the data they manipulate. Additionally, we have discussed selection and projection queries, the latter with two particular aggregation functions (sum and max); in [1] the authors study aggregate queries for "monoid" aggregate functions (including sum, count, min, max) for Probabilistic XML. Studying further operations such as value-based joins and projection with different aggregation functions, in the context of the different models presented here is an intriguing challenge.

Acknowledgments

This research was partially supported by the Israeli Science Foundation (ISF), the US-Israel Binational Science Foundation (BSF), the EU Project Mancoosi and the Israel Ministry of Science Eshkol grant.

6. REFERENCES

[1] S. Abiteboul, T.H. Hubert Chan, E. Kharlamov, W. Nutt, and P. Senellart. Aggregate queries for discrete and continuous probabilistic XML. In *ICDT*, 2010.

[2] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB J.*, 18(5), 2009.

[3] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *Proc. of EDBT*, 2006.

[4] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4), 2005.

[5] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In *Proc. of VLDB*, 2006.

[6] M. Benedikt, E. Kharlamov, D. Olteanu, and P. Senellart. Probabilistic XML via Markov chains. *PVLDB*, 3(1), 2010.

[7] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation.* Springer-Verlag, 1998.

[8] Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/ws-bpel/.

[9] T. Brazdil, A. Kucera, and O. Strazovsky. On the decidability of temporal properties of probabilistic pushdown automata. In *Proc. of STACS*, 2005.

[10] D. Chamberlin. Xquery: a query language for XML. In *Proc. of SIGMOD*, 2003.

[11] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4), 1995.

[12] T. Deng, W. Fan, L. Libkin, and Y. Wu. On the aggregation problem for synthesized web services. In *Proc. of ICDT*, 2010.

[13] D. Deutch. *Querying Web Applications Under Models of Uncertainty.* PhD thesis, Tel Aviv University, 2010. http://www.cs.tau.ac.il/ danielde/PhdThesis.pdf.

[14] D. Deutch and T. Milo. Querying structural and behavioral properties of business processes. In *Proc. of DBPL*, 2007.

[15] D. Deutch and T. Milo. Type inference and type checking for queries on execution traces. In *Proc. of VLDB*, 2008.

[16] D. Deutch and T. Milo. Evaluating top-k queries over business processes. In *Proc. of ICDE*, 2009.

[17] D. Deutch and T. Milo. Top-k projection queries for probabilistic business processes. In *Proc. of ICDT*, 2009.

[18] D. Deutch, T. Milo, N. Polyzotis, and T. Yam. Optimal top-k query evaluation for weighted business processes. In *Proc. of VLDB*, 2010.

[19] D. Deutch, T. Milo, and T. Yam. Goal-oriented web-site navigation for on-line shoppers. In *Proc. of VLDB*, 2009.

[20] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *Proc. of SIGMOD*, 2005.

[21] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. of PODS*, 2006.

[22] A. Deutsch and V. Vianu. Wave: Automatic verification of data-driven web services. *Data Eng. Bull.*, 31(3), 2008.

[23] P. Diniz. Increasing the accuracy of shape and safety analysis of pointer-based codes. In *LCPC*, 2003.

[24] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *Proc. of TACAS*, 2005.

[25] K. Etessami and M. Yannakakis. Recursive Markov Chains, stochastic grammars, and monotone systems of nonlinear equations. *JACM*, 56(1), 2009.

[26] C. Fritz, R. Hull, and J. Su. Automatic construction of simple artifact-based business processes. In *Proc. of ICDT*, 2009.

[27] M. R. Garey, R. L. Graham, and D. S. Johnson. Some np-complete geometric problems. In *Proc. of STOC*, 1976.

[28] R. Hull and J. Su. Tools for composite web services: a short overview. *SIGMOD Rec.*, 34(2), 2005.

[29] D. Jurafsky, C. Wooters, J. Segal, A. Stolcke, E. Fosler G. Tajchman, and N. Morgan. Using a stochastic context-free grammar as a language model for speech recognition. In *Proc. of ICASSP*, 1995.

[30] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Springer, 1976.

[31] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *Proc. of VLDB*, 2007.

[32] K. Lary and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algrithm. *Computer, Speech and Language*, 4:35–56, 1990.

[33] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems.* Springer-Verlag, 1992.

[34] T. Oates, S. Doshi, and F. Huang. Estimating maximum likelihood parameters for stochastic context-free graph grammars. In *Proc. of ILP*, 2003.

[35] T. Pavlidis. Linear and context-free graph grammars. *J. ACM*, 19(1), 1972.

[36] P. L. T. Pirolli and J. E. Pitkow. Distributions of surfers' paths through the world wide web: Empirical characterizations. *World Wide Web*, 2(1-2), 1999.

[37] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic XML data. In *Proc. of PODS*, 2007.

[38] L. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, Massachusetts Institute of Technology, 1974.