Querying Web-Based Applications Under Models of Uncertainty

Daniel Deutch

Supervised by Prof. Tova Milo
Tel Aviv University

ABSTRACT

Many businesses offer their services to customers via Web-based application interfaces. Reasoning about execution flows of such applications is extremely valuable for companies. Such reasoning must often operate under terms of uncertainty and partial information, due to partial tracing, effects of unknown external parameters, and more. The objectives of this research are (1) to define models for capturing Web application executions, with partial information and uncertainly of various flavors, (2) to design algorithms that allow for efficient reasoning over applications/execution traces under these models, and (3) to provide practical implementations that exploit these sound theoretical foundations for effective optimization of Web applications. We identify a restricted class of models that capture realistic scenarios, while allowing for an efficient query-based applications analysis. Hardness results indicate the necessity of such restricted models. We describe these results, highlight open problems, and consider directions for future research.

1. INTRODUCTION

A Business Process (BP for short) consists of some business activities undertaken by one or more organizations in pursuit of some particular goal. Customers (and other businesses) often interact with such BPs via Web-based application interfaces, which are extremely popular nowadays. The flow of such applications depends on many variables whose values are known only at run-time. Among them one may find user choices, servers availability, response time and more. To allow for applications analysis, it is a common practice to *trace* the execution flow, logging the performed activities along with their relative order and causality relationship. Reasoning about execution traces of such applications is extremely valuable for companies: it can be used to optimize business processes, employ targeted advertisements, reduce operational costs, and ultimately increase competitiveness. Such reasoning often operates in an environment that induces partial information and uncertainty of various flavors. First, the execution traces recorded for a Web application often contain only partial information on the activities that were performed at run time. This may be due to confidentiality, lack of storage space, etc. Second, even in the presence of fully detailed traces of the past executions, prediction of the behavior of future executions should naturally consider uncertainty. This is because executions often depend on unknown external parameters, such as users behavior, interaction with other applications, servers response time, etc. To enable such reasoning, the possible/typical execution flows of applications must be first characterized and then effectively analyzed. We focus on two complementing sorts of characterization, detailed below.

Traces structure. Schema and type information has proved to be extremely useful for the management of semi-structured and XML data. Knowledge about the typical structure (shape) of data items allows for intuitive query formulation, optimized query processing, minimization of run time errors, and more [19, 4]. Such knowledge is commonly referred to as *type information*. Process execution traces can be abstractly viewed as a particular class of semi-structured data having the shape of Directed Acyclic Graphs (DAGs), which describe the execution flow. [3] showed that type information is also critical for (optimized) analysis of execution traces.

While formal process specifications (e.g. in the style dictated by the BPEL standard [5]) provide knowledge about the possible shape of execution traces, it turns out that a much more careful characterization of the traces shape is required in practice. First, as mentioned above, the execution traces recorded for Web applications often contain only partial information on the activities that were performed at run time (namely only part of what is given in the formal process specification is actually recorded). Second, when execution traces are queried, the process specifications provide type information only about the *input* traces, but not about the *output* sub-traces selected by queries. To address this, we study, for the first time, the management of type information for process execution traces. Specifically, we consider here type inference and type checking for queries over BP execution traces. The queries that we consider select portions of the traces that are of interest to the user; the types describe the possible shape of the execution traces in the input/output of the query. We formally define and characterize here three common classes of BP execution tracing systems that vary in the amount of information that they record on the run, and consider their respective notions of type inference and type checking. We study the complex-

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

ity of the two problems for query languages of varying expressive power and present efficient type inference/checking algorithms where possible. Our analysis aims to provide a complete picture of which combinations of trace classes and query features lead to PTIME effective algorithms and which to NP-complete/undecidable problems.

Traces likelihood. A second characterization of interest is by *likelihood.* The execution course of a BP depends on many variables, such as users behavior, interaction with other applications and servers response time, whose actual value is known only at run time. However, examining a set of execution traces, one will often observe that some execution traces (or patterns thereof) are more likely than others. Identifying the top-k most likely traces (satisfying certain query criteria), is crucial for various applications, such as adjustments of Web-sites design to fit the needs of certain user groups, personalization of on-line advertisements, enhancements to business logic etc. To allow for top-k traces selection, one first needs a model describing all the possible executions of the BP, as well as their likelihood. Second, given such a model and a query describing a family of execution traces (usage patterns) of interest, an efficient mechanism to identify the top-k qualifying traces is required. Much efforts have been directed recently to addressing the first issue, namely the inference of a proper BP model out a given set of execution traces, using a variety of data mining techniques (see related work). The second issue, namely efficient querying and analysis of the BP model, has been less successfully addressed so far. Our work aims at providing a set of algorithms that address this problem, under various settings. These algorithms will be a basis for practical implementations that exploit the sound theoretical foundations for effective optimization of Web applications.

In the following sections we give an intuitive, *informal* description of our models and the results obtained so far. The formal definitions and the detailed theorems and algorithms appear in [9, 11].

2. PRELIMINARIES

We start by reviewing (informally) the main concepts that stand at the center of our research. In this section, we introduce a basic form of each concept; their extended variants that capture uncertainty are discussed in section 3. While the models that we use for describing BP specifications and queries are an abstraction of those considered in [2, 3], they are abstract, conforming to various formalisms.

Business Processes. We use below the term Business Process (BP) to describe the essence of the (Web) applications that we consider here. The recent BPEL standard (Business Process Execution Language [5]) allows to describe the full operational logic and execution flow of BPs. A BPEL *specification* describes a process as a nested DAG (Directed Acyclic Graph) consisting of activities (nodes), and links (edges) between them. Links detail the execution order of the activities. Activities may be either *atomic* or *compound.* In the latter case, each of their possible internal structures (called *implementation*) is also detailed as a DAG, leading to the nested structure. In each DAG, a unique node is marked as `start` (`end`) node.

EXAMPLE 2.1. *A schematic graphical example for a BP specification of a Web-based travel agency is depicted in Figure 1. Each activity is represented here by a pair of nodes, the first (having darker background) standing as the activity's activation point and the second as its completion point. The BP starts at the compound* `Trip` *activity. Then, the user may choose between reserving a regular trip and a luxurious one. The first choice leads to the implementation whose flow is depicted in the leftmost big bubble. The second choice leads to the* `Luxury` *compound activity, whose internal flow (implementation) is depicted in the rightmost big bubble. In the two implementations, the user may search,* in parallel *(e.g. by opening new browser windows), for hotels and flights (regular or luxury). In each implementation, the user is directed to a billing system, represented by the* `Credit1` *and* `Credit2` *activities (for regular and luxury reservations, respectively).*

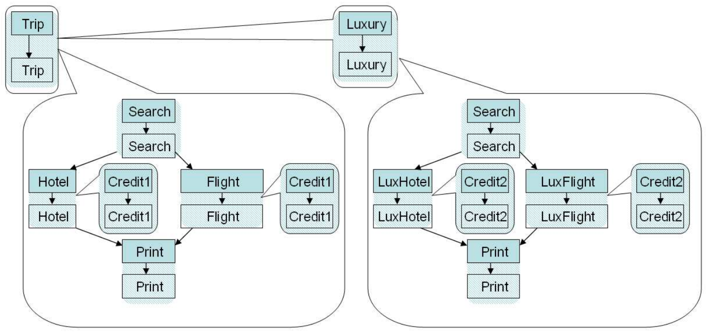

Figure 1: Business Process

Execution Traces. An *instance* of a BPEL specification is an actual running process that follows the logic described in the specification. BP Management Systems allow to trace instance executions. An execution trace can be abstractly viewed as a (nested) DAG that contains nodes representing the activation (start) and completion (end) events of activities, and edges that describe their flow. For a compound activity, the events corresponding to its internal flow are recorded between its activation and completion events, and are connected to them using distinctly marked *zoom-in edges.* The nesting of DAGs in the trace follows naturally from the BP nested structure.

EXAMPLE 2.2. *Consider the trace depicted in Figure 2(a).* zoom-in edges *are depicted as dashed arrows, and following them reveals internal traces of the corresponding compound activities. For example, zooming into* `Trip` *reveals that a* `Search` *was performed, then the corresponding hotel and flight were reserved (in parallel), by the* `Hotel` *and* `Flight` *activities, resp., and a confirmation was printed.*

Queries. We use an abstraction of the query language suggested in [3], where queries are defined using *execution patterns*, whose structure is similar to that of execution traces. Some of the pattern's edges may be marked as transitive, seeking for a path connecting the edge end-nodes, rather than a single edge; similarly, composite nodes may be marked as *transitive*, seeking for *possibly indirect* implementations.

To evaluate a query, we search for occurrences of the execution pattern in the execution traces, represented by *embeddings.* An embedding is a homomorphism from all nodes

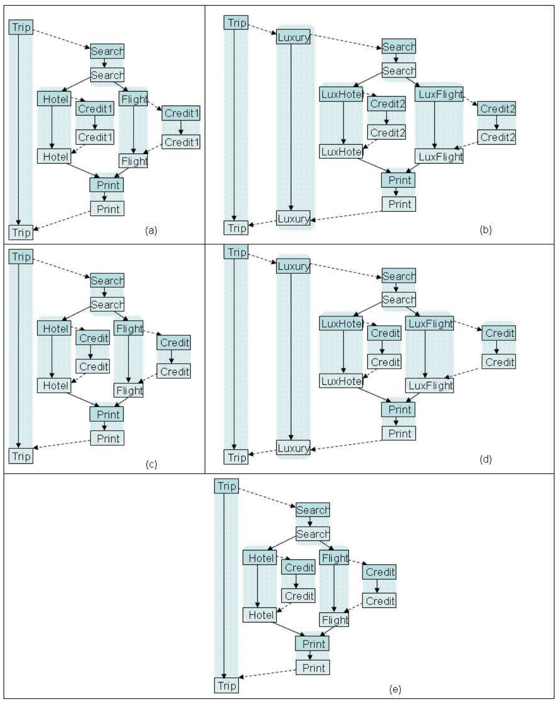
Figure 2: Execution traces

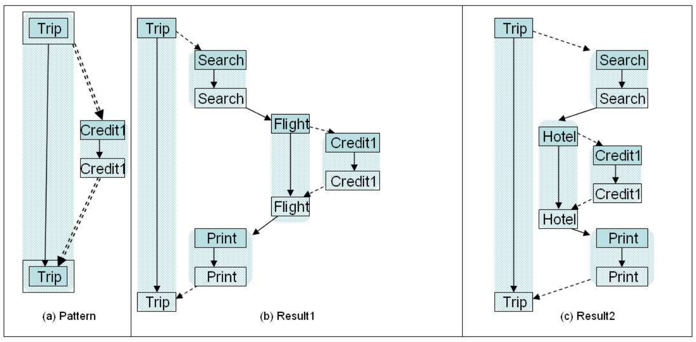

Figure 3: A query and its results

and edges of the pattern to some nodes and edges of the trace, such that for matched nodes, their activity names coincide, and composite (atomic) nodes are mapped to composite (atomic) nodes. Non-transitive edges (both regular and zoom-in) of the query are mapped to corresponding edges of the trace, such that the end-points in the pattern are mapped to the corresponding end-points in the trace. Transitive edges may be mapped to paths (containing flow or zoom-in edges), and implementations of transitive query nodes may be mapped to indirect implementations of the corresponding nodes in the specification.

EXAMPLE 2.3. *An example execution pattern is depicted in Fig. 3 (a). It zooms-in transitively into* `Trip`*, searching for a* `Credit1` *activity. Double-bounded nodes (double-lined edges) denote transitive nodes (resp. edges). Fig. 3(b) and (c) depict its two possible matches in the trace of Fig. 2(a).*

Types. A family of BP execution traces is captured by the notion of *type*. A type consists of (1) a BP specification and (2) a tracing system, that records activities of the BP execution. In the simplest case, *all* activities performed by the BP are recorded. More general cases, where only partial information is recorded, will be considered in the following section. For queries over BP execution traces we will consider both the *input type*, for which the BP specification is readily available, as well as the *output type* that captures sub-traces selected by queries, and for which type information will be inferred.

3. MODELS OF UNCERTAINTY

We next extend our model to account for uncertainty caused by partial tracing, as well as uncertainty caused by external events. We present the corresponding problems, our proposed solutions, and their practical implications.

3.1 Partial Tracing

We present a model for partial tracing, then list our initial results for *Type Inference* and *Type Checking* in these settings.

Model Loggers that generate execution traces of BPs may vary in the amount of information that they record on runs. To account for that, we distinguish three families of execution trace types, with increasing flexibility (and decreasing amount of guaranteed information), as follows: *Naive tracing* provides a complete record of tracing provides a complete record of the activation/completion events of all activities, along with the corresponding activities names; *Semi-Naive tracing* provides a record of all activation/ completion events, but possibly with only partial information about their corresponding activities names; and *Selective* tracing provides only a record of some subset of the activation/completion events, possibly reported with only partial information about their corresponding activities names.

Next, we present examples for the different sorts of tracing systems, and the obtained traces.

EXAMPLE 3.1. *Figures 2 (a) and (b), described above, present examples for* naive *traces of the BP in Figure 1. Note that these traces disclose that a different billing system is invoked for luxury and regular reservations. To avoid such disclosure, we may rename the logged activities, so that the traces contain a generic activity name such as* `Credit`*, instead of the* `Credit1` *and* `Credit2` *names. This is captured by the notion of* semi-naive *tracing system, represented by a* renaming function *over the activities names. Figures 2 (c) and (d) present semi-naive traces of our BP, obtained from the executions that led to Figures 2 (a) and (b) resp.*

In some cases, a more selective tracing is used, where the occurrence of some activities is not recorded at all, due to confidentiality, storage constraints etc. A selective tracing system is represented by a renaming function and a deletion set; the record of all occurrences of activities names that appear in the deletion set is omitted. For instance, if our travel agency wishes to keep as a secret the fact that reservations of different types are treated differently, then not only the credit checks activities need to be re-labeled, but also the `LuxHotel` *and the* `LuxFlight` *activities, and the record of the* `Luxury` *activity should be omitted altogether. Figure 2(e) is the selective trace obtained from the naive trace in Figure 2 (b), as well as by (a), by applying such renaming and deletion operations. Thus, the goal of secrecy is achieved.*

Types Revisited We have mentioned above that a *type* for a set T of traces consists of (a) a Business Process specification and (b) a tracing system, such that T is exactly the set of traces logged for all possible executions of the process. We have already defined the notion of business processes, and we may now formally formulate the notion of a tracing system: it consists of (1) a *renaming function* over the activities names and (2) a deletion set, which is a subset of the activities names set. The semantics is that run-time occurrences of activities are recorded as their image under the renaming function; and occurrences of activities whose names appear in the deletion set are omitted from the execution log altogether. If the renaming function is the identify function and the deletion set is empty, the tracing system is naive; if only the latter holds, the tracing system is semi-naive; otherwise selective.

Our results We study the problems of Type Checking and Type Inference for queries over execution traces, for all cases of tracing systems. The two problems are practically well-motivated, as querying execution traces repositories is often done in two steps: the repository is first queried to select portions of the traces that are of particular interest. Then, these serve as input to a finer analysis that further queries and mines the sub-traces to derive critical business information [22]. Not surprisingly, type information, i.e., knowledge about the possible structure of the queried (sub-)traces, is valuable for query optimization [3]. Its role is analogous to that of XML schema for XML query optimization: it allows to eliminate redundant computations and simplify query evaluation. Such type information is readily available, as the BPEL specification, for the original traces. However, this information is not available for the sub-traces selected by queries. *We develop efficient algorithms for deriving this type information.* At another level, when the analysis tool expects data of particular type, we would like to guarantee that the sub-traces selected by the queries conform to the required type. *Such type checking is a second problem that we study.* We examine the two problems in presence of naive, semi-naive and selective tracing systems, for both the input and output types. Our results follow. For space constraints, we may not give here the exact proofs for our results, but rather explain the intuition behind them; the exact theorems and proofs appear in [11].

Type Inference. In the type inference problem we are given an input type and a query over its traces, and our goal is to infer an output type that represents only the (sub-) traces that conform to the query. The problem is naturally parameterized by the tracing systems of the input and output type. We consider all possible variants:

- If the output type is restricted to a naive tracing system, we show that type inference may not be possible.
- If the output type is restricted to a semi-naive tracing system, we give an algorithm for type inference. However, we show that, inevitably, the size of the output type may be exponential in the size of the input type and query. This holds even if the input type is restricted to a naive tracing system.
- If a further more flexible, selective tracing system is allowed for the output type, we provide an algorithm that performs type inference, whose time complexity is polynomial in the size of the input type, with the exponent determined by the query size. This holds even if the input type contains a selective tracing system.

Intuition

- To observe that naive tracing system may be insufficient for capturing exactly the traces set selected by a query, consider a BP specification s whose flow contains two consecutive nodes labeled by the compound activity name a. a bears two different implementations, one containing an activity name b, and the second containing c. Obviously, the set of possible execution traces corresponds to all combinations of b / c followed by b / c. Now consider a query that requires a choice of b for the first a-labeled activity, and a choice of c for the second b-labeled activity. It is easy to see that no BP specification may have the unique trace corresponding to the query as its only naive trace
- In contrast, if the output type may use a semi-naive tracing system, type inference is possible. For the above example, the new BP will contain two activity names a_1 and a_2, the first having b as its single implementation and the second having c as its single implementation. The renaming function will then map both a_1 and a_2 to a.

  In general, the type inference algorithm follows the lines of the query evaluation algorithm in [10]. The algorithm generates an "intersection" between the original BP specification and the query. The pattern specified in the query is embedded within the specification in a top-down manner; start by mapping the query root to the specification root, and then map recursively each implementation of a mapped query node, to some implementation within the specification. As for transitive nodes, the corresponding implementation in the specification does not have to be a direct one, thus we consider all splits of the query, allocating a sub-query to each compound activity name in the newly created specification. This allocation is done by renaming the activity names into pairs of [activity name, sub-query], and thus the semi-naive tracing is required.

  The exponential blow-up in the query size is due to the need of considering all query splits. The blow-up in the specification size is due to *transitive edges*, as the number of possible paths matching to such edge may be exponential in the specification size.
- *Selective tracing* allows to avoid the exponential blow-up in the specification size caused by matchings transitive edges. The selective tracing is exploited to form a *regular grammar*, succinctly capturing all paths to which the transitive edge was matched. The overall complexity is thus polynomial in the specification size, with the exponent determined by the query size (all splits of the query still need to be considered).

Type Checking. In the type checking problem we are also given, in addition to the input type and query, a target type. Our goal is to verify whether the (sub-) traces that conform to the query, out of those of the input type, also conform to the target type. Again, we consider several variants, as follows:

- If the input and target types are restricted to naive or semi-naive tracing systems, we give an EXPTIME algorithm for type checking, and show NP-hardness of the problem.

- We show that type checking is generally undecidable if the input and target types may use selective tracing.

Intuition

- The EXPTIME algorithm is rather complex, and is derived from the algorithm for intersecting parenthesis string grammars [18]. The general idea is to define an auxiliary class of *deterministic business processes*, where every logged execution trace uniquely defines the execution that occurred in practice; then we show that intersection of such processes is possible, and finally we show that every business process may be transformed to a deterministic one. The details may be found in [11].

- The undecidability proof is by reduction from the problem of testing containment of context free (string) languages, known to be undecidable.

Practical Implications. Our results on type analysis have two main practical implications. First, concerning type inference, we signal the class of *selective trace types* as an "ideal" type system for BP traces, allowing both flexible description of the BP traces as well as efficient type inference. Second, we have shown the hardness of type checking, which motivates identifying further practical restrictions that allow efficient type checking.

3.2 Traces Likelihood

We start by explaining our model for external events that affect the course of execution, then proceed to the results.

Model We first extend the definition of BP specifications to support external events and likelihood. Then we define a ranking over traces, based on their likelihood, and finally adjust the query language to account for TOP-K queries.

Specification. Recall that each compound activity may have different possible implementations, corresponding to different user choices, variable values, server availability etc. To capture these notions, we attach logical formulas (over user choices, variable values, etc.) *guarding* each implementation.

Probabilities. In practice, some guarding formulas truth values are more likely than others. Moreover, the likelihood of any particular choice varies at different points of the run and may depend on the course of the run so far and on previously made choices. To account for that, we use a probability distribution function δ that, given a description of an execution course up to a given point, and a guarding formula f, determines the probability that f holds. The likelihood of each execution trace is defined based on this distribution function. We focus on three common classes of distribution functions, as follows: *memory-less* where all formulas are independent, *fixed-choice*, where the truth value of each formula stays constant over time, and *bounded-memory*, where the truth value of each formula f at any point of execution depends (at most) on truth values assigned to the k most recent occurrences of each formula, for some constant k.

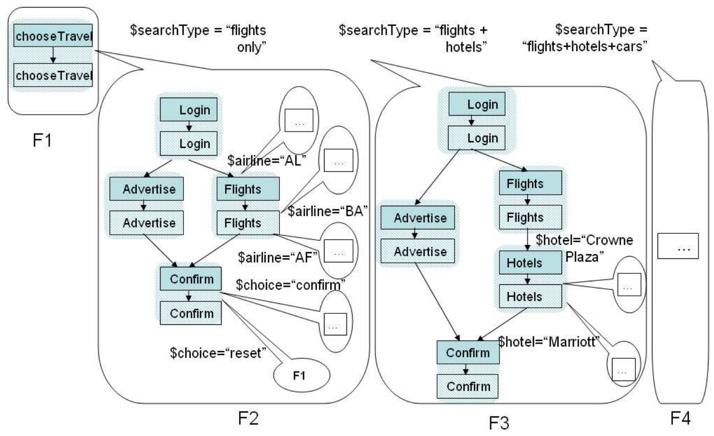

Figure 4: Probabilistic BP

$\$searchType$	P($\$searchType$)
"flights only"	0.65
"flights + hotels"	0.25
"flights + hotels + cars"	0.1

$P(\$Airline \mid \$searchType)$	"flights only"	"flights +hotels"	"flights +hotels+cars"
"BA"	0.7	0.1	0.6
"AF"	0.1	0.7	0.3
"AL"	0.2	0.2	0.1

$P(\$Hotel \mid \$Airline)$	"BA"	"AF"	"AL"
"Marriott"	0.7	0.6	0.1
"HolidayInn"	0.05	0.1	0.7
"CrownePlaza"	0.25	0.3	0.2

$P(\$choice \mid \$AirLine)$	"BA"	"AF"	"AL"
"reset"	0.5	0.4	0.8
"confirm"	0.5	0.6	0.2

Table 1: Formulas Distribution

EXAMPLE 3.2. *Figure 4 provides a schematic description of the Travelocity [24] application. Each activity roughly corresponds to a Web-page in the application. Implementations of compound activities are now guarded by* guarding formulas*, which are boolean formulas over parameters such as user choices (e.g. the choice of* \$searchType*, or of* \$airline*, where "BA" stands for British airways, "AF" for Air France, "AL" for Aer Lingus). At run-time, one implementation will be chosen for each compound activity occurrence, determined by the formulas' truth values. Table 1 depicts a bounded-memory distribution over these formulas. The conditional probability of each formula is given succinctly, where each formula is conditioned only on a subset of other formulas; this subset is sufficient for computation of the formula's probability, independently of other formulas. Here, the top-3 most typical execution flows are the following: the first is the one obtained where the user searches only for flights, booking a BA flight (its likelihood is 0.2275). The second is one where the user chooses to search for flights and hotels, booking an AF flight and a Marriott hotel (0.063). Finally, the third is one where the user searches only for flights, choosing BA as airline, but is not satisfied and retries, this time reserving only a BA flight (0.05175625). Note that the specification structure dictates the sequence of guarding formulas values that correspond to each trace, and thus affects the computation of likelihood.*

The analysis in the above example suggests that hotel deals presented to British Airways flyers are not appealing enough and need to be improved. We next explain how Top-k querying can assist with that.

Query Language. We adjust the query language presented in section 2, to support queries retrieving only the TOP-

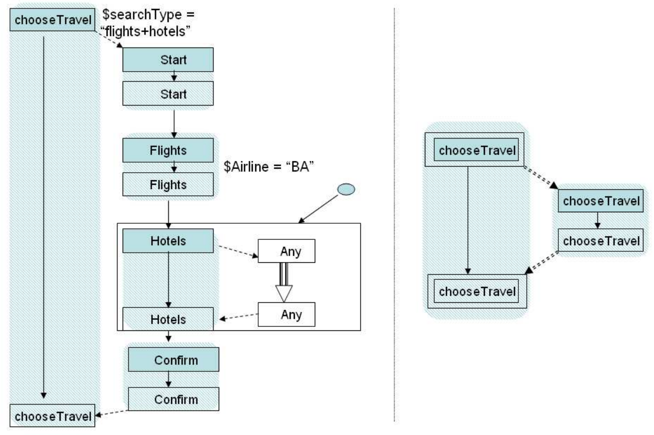

Figure 5: (a) Query 1 (b) Query 2

K most likely matched traces. Examples for such queries follow.

Example 3.3. *The query depicted in Figure 5(a) focuses on British Airways fliers that also reserved some hotel, and retrieves the top-k hotel-related activity flows of such traces. The top-1 match for the query indicates that British Airways travelers prefer the Marriott, which should thus be further promoted. Fig. 5(b) depicts a query that focuses on executions where the search ended unsuccessfully, with the users "resetting" and (recursively) restarting the search. Identifying the most common such flows may provide insight on main reasons for users dissatisfaction.*

Our results Finding the TOP-K traces is called the TOP-K-TRACES problem (BEST-TRACE, for the decision problem), and its variant that considers also a query is called TOP-K-MATCHES (BEST-MATCH).

- Given a specification s, accompanied by a memory-less or fixed-choice distribution, we present and implement an algorithm that computes TOP-K-TRACES in time polynomial in $\mid s \mid$. Our algorithm computes a type representing only the TOP-K (relevant) traces. Given also a query q, we may compute TOP-K-MATCHES in time polynomial in $\mid s \mid$ with the exponent determined by $\mid q \mid$. We also show that BEST-MATCH is NP-complete in $\mid q \mid$.
- For bounded-memory distributions, BEST-TRACE (and thus also BEST-MATCH) is NP-complete in the specification size. However, we present and implement algorithms for TOP-K-TRACES and TOP-K-MATCHES that perform well on practical cases.
- We show that these results may not be extended to general distributions, as for such distributions, BEST-TRACE (BEST-MATCH) is undecidable.

Practical Implications. We have implemented our algorithms, and tested their performance on synthetic data. Our results so far indicate that analysis of fairly large specifications, with thousands of activities, may be performed within seconds. As a further goal, we intend to apply our results to real-life objectives, as described below.

4. FUTURE WORK

We have summarized above our main results. In this section, we review our main goals for further research.

Integration. So far, we have considered separately our two models for uncertainty. Thus, the likelihood analysis algorithms assumed that the tracing system is *Naive*, revealing all information on logged execution traces. Naturally, we intend to extend TOP-K-TRACES and TOP-K-MATCHES to settings of semi-naive and selective tracing.

Open Problems. Clearly, there exist related problems that currently remain open. We list some of these next.

- **Efficient Type Checking.** We have given EXPTIME algorithm for type checking in presence of semi-naive tracing systems, and have shown undecidability for selective tracing systems. We are interested in identifying restrictions, either over the tracing systems or over the specification, allowing PTIME algorithms.
- **Models for Distribution.** Our algorithms are all generic, and use an oracle that allows computation of probabilities and (conditional) independencies. We intend to examine appropriate implementations for this oracle; preliminary results indicate that dynamic bayesian networks, of varying expressivity, may serve as appropriate models for this need.
- **Specification Inference.** In practice, there are many cases in which the distribution over the specification guarding formulas, or even the specification structure itself, are unknown, and should be mined from a given set of sampled traces. Incorporation of such inference engine is one of our challenges for future research.
- **Optimizations.** We intend to enhance our optimizations that allow practically efficient run-time.

Practical Applications. We intend to construct a system that demonstrates practical applications of our algorithms in the context of real-life commercial Web-sites. *Web sites design* may be adjusted to fit navigation patterns of specific classes of users, either statically or dynamically; *On-line advertisements* may be injected according to common patterns of co-occurring choices. Given the structural nature of the analysis, we may point out not only the content of personalized advertisements, but also specific locations within the Web-sites, where advertisements will yield optimal profit; and more.

5. RELATED WORK

We give in this section a review of related work, highlighting the relative contributions of our results. Due to space constraints we give here only a limited subset of the related results. See [11, 9] for further review.

Types. Type checking and type inference are well studied problems in functional programming languages [20]; as pointed out in [6], this analysis is valuable for database queries as well. Type inference and type checking were also considered extensively in the context of XML. [19] showed that for XML selection queries, type checking can be performed in time complexity equal to or lower than type inference (depending on the XML types/queries being considered). Compare to our setting (baring the obvious distinction of having nested DAGs instead of flat trees), where type checking may be harder than type inference.

Probabilistic data. Probabilistic Databases (PDBs) [8], Probabilistic XML [1] and Probabilistic Relational Models (PRMs) [15] allow representation of uncertain information. Many works on PDBs assume independencies between the data elements. Dependencies are considered in PRMs and in extensions of PDBs [23], however they do not capture the dynamic nature of flow and the possibly unbounded number of recursive (possibly dependent) activity invocations. In terms of *possible worlds semantics*, the number of worlds in our model is infinite, and thus materializing all worlds is not only costly (as in PDBs), but impossible. Extensions of PRMs to a dynamic setting [21] do not allow for efficient query evaluation.

Grammars. There is a tight connection between the classes of traces studied here and corresponding classical classes of *string* languages, represented by bracketed, parenthesis and context-free grammars [16]. There is also a close connection between selective trace types and context free *graph* languages [7]. However, to the best of our knowledge, no model equivalent to naive and semi-naive tracing was studied in this context. In terms of query languages, most of the work on context free graph grammars is concerned with formal logic, and specifically First and (Monadic) Second Order Logic (MSO). Our query language is expressible in MSO, but is restricted enough to allow efficient evaluation, absent from [7].

Verification. A variety of formalisms for (probabilistic) process specifications exist in the literature. Among these, we mention Hidden Markov Models (HMMs) [13], (Probabilistic) Recursive State Machines (PRSMs) [15], and Stochastic Context Free (Graph) Grammars (SCFG, SCFGG). While HMMs extend Finite State Machines, PRSMs and SCFGGs describe nested structures similar to our model. However, they assume independencies between probabilistic events. As noted in [10], our query language allows representation of properties inexpressible in temporal logics typically considered in verification, as it is not bisimulation-invariant. Also note the unique concepts of trace types defined here, and the efficiency of TOP-K query evaluation, both absent from verification works on probabilistic processes.

Data Mining. Finally, we mention a complementary line of tools that use data mining techniques to autonomously (that is, without queries, and generally without knowledge of the specification structure itself) analyze a set of traces. Typically, these works aim at finding, autonomously, "interesting" patterns within the given logs, or inferring an approximation of the specification. Among these we mention Web mining [14], Business Process Mining [17], and OLAP-style tools [12].

6. CONCLUSION

The research described in this paper focuses on Web applications, aiming to develop tools for their analysis, in settings of partial information and uncertainty. Initial results indicate that our models are suitable for reasoning over such specifications, being concise, intuitive, and allowing efficient analysis. Based on these models, we provide efficient and practical methods for analyzing and querying specifications under terms of uncertainty; experimental results, as well as theoretical analysis, indicate the efficiency of our methods. We believe that our models and results stand as firm foundations for further research; we intend to enhance and optimize our methods, solve problems that remain open, and finally implement our algorithms to allow a representation and querying of real-life Web applications, under realistic settings.

7. REFERENCES

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

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

[3] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *Proc. of VLDB*, 2007.

[4] V. Benzaken, G. Castagna, D. Colazzo, and K. Nguyen. Type-based xml projection. In *Proc. of VLDB*, 2006.

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

[6] J. Bussche, D. Gucht, and S. Vansummeren. A crash course on database queries. In *Proc. of PODS*, 2007.

[7] B. Courcelle. The monadic second-order logic of graphs. *Inf. Comput.*, 85(1), 1990.

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

[9] D. Deutch and T. Milo. Querying probabilistic execution traces. http://www.cs.tau.ac.il/~danielde/ProbTracesFull.pdf.

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

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

[12] J. Eder, G. E. Olivotto, and W. Gruber. A data warehouse for workflow logs. In *Proc. of EDCIS*, 2002.

[13] Y. Ephraim and N. Merhav. Hidden markov processes. *IEEE Trans. Inf. Theory*, 48(6), 2002.

[14] F. M. Facca and P. L. Lanzi. Mining interesting knowledge from weblogs: a survey. *Data Knowl. Eng.*, 53(3), 2005.

[15] N. Friedman, L. Getoor, D. Koller, and A.Pfeffer. Learning probabilistic relational models. In *Proc. of IJCAI*, 1999.

[16] S. Ginsburg and M. Harrison. Bracketed context-free languages. *J. Computer and System Sciences*, 1, 1967.

[17] D. Grigori, Fabio Casati, M. Castellanos, M.Sayal U .Dayal, and M. Shan. Business process intelligence. *Computers in Industry*, 53, 2004.

[18] R. McNaughton. Parenthesis grammars. *J. ACM*, 14(3), 1967.

[19] T. Milo and D. Suciu. Type inference for queries on semistructured data. In *Proc. of PODS*, 1999.

[20] J. C. Mitchell. *Foundations for Programming Languages.* MIT Press, 1996.

[21] S. Sanghai, P. Domingos, and D. Weld. Dynamic probabilistic relational models. In *IJCAI*, 2003.

[22] D. M. Sayal, F. Casati, U. Dayal, and M. Shan. Business Process Cockpit. In *Proc. of VLDB*, 2002.

[23] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proc. of ICDE*, 2007.

[24] Travelocity web-site. http://www.travelocity.com.