A Provenance Framework for Data-Dependent Process Analysis *

Daniel Deutch
Tel Aviv University

Yuval Moskovitch
Tel Aviv University

Val Tannen
University of Pennsylvania

ABSTRACT

A data-dependent process (DDP) models an application whose control flow is guided by a finite state machine, as well as by the state of an underlying database. DDPs are commonly found e.g., in e-commerce. In this paper we develop a framework supporting the use of provenance in static (temporal) analysis of possible DDP executions. Using provenance support, analysts can interactively test and explore the effect of hypothetical modifications to a DDP's state machine and/or to the underlying database. They can also extend the analysis to incorporate the propagation of annotations from meta-domains of interest, e.g., cost or access privileges.

Toward this goal we note that the framework of semiring-based provenance was proven highly effective in fulfilling similar needs in the context of database queries. In this paper we consider novel constructions that generalize the semiring approach to the context of DDP analysis. These constructions address two *interacting* new challenges: (1) to combine provenance annotations for both information that resides in the database and information about external inputs (e.g., user choices), and (2) to finitely capture infinite process executions. We analyze our solution from theoretical and experimental perspectives, proving its effectiveness.

1. INTRODUCTION

Complex software applications whose control flow is dependent on an underlying database as well as on external input are commonly found in a wide range of domains. For example, E-commerce applications rely on a database for management of products, orders etc., affecting the possible execution of the application and thus its interaction with potential users. We model such applications through *data-dependent processes (DDPs)* which are finite state machines (FSM) whose transitions are guided by both external input and by the result of internal database queries. Due to the complexity and importance of such applications, it is a common practice to perform an automatic *(static) analysis* of properties related to anticipated application executions, using formulas in temporal logic languages.

As a simple example, consider a typical E-commerce application where users can navigate through a selection of products (classified through categories and sub-categories), and proposed discount deals. The underlying process semantics can be captured via an FSM whose states are associated with web-pages and/or properties defined by the applications' business logic. The transitions between states can be governed both by user clicks or text input and by queries on the product database. In assessing such an application an analyst may be interested in questions such as "can a user view a particular category without being a club member?", or "what is the minimum number of clicks allowing a user to view the daily deals being offered?".

To obtain a better understanding of the application, an analyst may want to do more than pose static analysis questions with respect to its current specification. Indeed, she may also want to test and explore the effect on analysis results of *hypothetical scenarios*. Such scenarios may involve different modifications to the business logic (e.g., the application's link structure) or to the underlying product data. In realistic cases, there may be many possible scenarios and their *combination* that are of interest. An analyst would like to interactively explore the different combinations, refining them according to the analysis results (e.g., gradually removing links and observing the effect). In the context of SQL query answers it was shown in [12] that provenance annotations enable such interactive exploration under combinations of hypothetical scenarios, a technique that we refer to as *provisioning*. This technique changes the queries so they compute an answer (a *provisioned autonomous representation*) that incorporates compactly provenance annotations corresponding to the hypothetical scenarios.

In addition, we consider analysis tasks that involve the propagation of *annotations* from meta-domains such as cost (e.g., number of clicks), trust (e.g., confidence scores), or security (e.g., access control/clearance levels). Here we are interested in starting with annotations that capture assumptions about the underlying database tuples or external inputs. We then propagate them to annotations of analysis results. In other contexts (various query languages) it was shown that it suffices to compute provenance and then evaluate it (*specialize* it) in specific meta-domains [23, 17, 4].

*This research was partially supported by the Israeli Ministry of Science, the Israeli Science Foundation, the National Science Foundation (NSF IIS 1217798) and the US-Israel Binational Science Foundation.

Proceedings of the VLDB Endowment, Vol. 7, No. 6

Given these important applications, the goal of this paper is *to study provenance management for the analysis of data-dependent processes*. We start from the generic framework of *semiring-based provenance* introduced in [23] and extend it to DDPs and temporal logic analysis.

While analysis of Data-Centric Processes was studied in e.g., [9, 14, 18], the focus on provenance and its applications (provisioned temporal analysis and specialization) distinguish the present work from that research.

Several conceptual and technical challenges need to be addressed in such a study. We review here these challenges and our contributions in addressing them.

Process model and analysis formalism (Section 2). Our first step is to define the DDP model. A DDP is specified by an FSM whose transitions are either guarded by a yes/no query on a static underlying database, or otherwise considered as non-deterministic. Intuitively, non-deterministic transitions correspond to *external effects* such as user choices, interacting applications, etc. For boolean query guards we allow queries in positive relational algebra with aggregates. For analysis we follow common practice and use *Linear Temporal Logic* (LTL) [29]. An analysis task may be represented by an LTL formula, which consists of (1) "atomic" predicates that may be evaluated with respect to process states (e.g. state names), (2) standard logical connectives (and,or,not) and (3) "temporal operators" that allow to express required relationships between truth values of predicates throughout an execution (e.g. a state S_1 must appear before S_2). We focus here on analysis of *finite* possible DDP executions, following the perspective used in workflow provenance; there may still be *infinitely many* such executions if the process logic involves loops.

Provenance Model (Section 3). We further develop a *provenance model* for the result of LTL formulas with respect to a DDP. The development of the provenance model must account in particular for the different kinds of transition choices (depending on queries on the data or on external effects) and for the possibly infinitely many executions. Thus, the semiring-based model [23] cannot be used as is, and a novel construction is required. We start by allowing the separate annotation of data tuples and external choices, using two different semirings. This separation allows us e.g. to simultaneously do provisioning for data tuples and provenance specialization for external choices. For example "what is the minimum number of clicks to view the daily deals, if the database is modified in a particular way".

Then, we propose a novel construction that allows capturing *pairs of provenance annotations*. The paired components correspond to external choices and to query answers on the underlying database. A novel construction is needed because completely separating the two kinds of provenance is not a good idea. In semiring provenance alternatives are modeled by addition and joint use is modeled by multiplication [23]. The semantics of DDPs involves alternative paths. If we accumulate the database provenance separately from the external effects provenance we lose track of when both happen along the *same* path. Therefore instead of taking the cartesian product of the two semirings, we need to factor by some algebraic congruences on pairs. We also need to accommodate the tracking of provenance over possibly infinitely many different executions. To this end we construct a *tensor product* of two semirings that manipulates *infinite* bags of pairs before factoring through the desired algebraic congruences. This allows us to support provenance for infinitely many paths, to greatly simplify the resulting expressions and to use the provenance expression through semiring homomorphisms (see [23]).

Finally, we show with examples that multiplication and addition in the resulting structure can be interpreted as *joint* and *alternative* use (resp.). This interpretation justifies our definition of provenance for LTL formulas, namely, as the (possibly infinite) sum over all the paths that realize the formula; along each path we compute provenance as a multiplication of provenance of the individual transitions.

Provenance Computation and Usage (Section 4). Given the DDP and provenance models we address the computation of provenance of an LTL formula with respect to a given DDP. We show how to generate a (finitely described) expression in the tensor product structure that captures the provenance. We further analyze the complexity of computing such expressions as well as their possible sizes. Then we consider the *commutation with homomorphism* property, which was proven for various database query languages in [23, 17, 4]. The property is essential for the soundness of applying provenance to provisioning and to specialization in meta-domains of interest. This is because the provenance expressions can be built using indeterminate parameters (variables) as input annotations and then both hypothetical scenarios and meta-domain assumptions correspond to valuations of these parameters in specific semirings. These valuation determine homomorphisms through which the evaluation of provenance expressions is done and the correctness of our method relies on commutation with homomorphisms. We prove this property indeed holds for our construction.

Prototype Implementation (Section 5). We have implemented our model and algorithms in the context of a system prototype called `PROPOLIS` (PROvisioned data-dependent PrOcess anaLysIS) [13]. `PROPOLIS` allows analysts to define, in addition to an LTL formula, *annotations* (from e.g. cost, trust, and access-control meta-domains) on the process model. This captures *hypothetical scenarios* by *parameterizing* the process specification (its logical flow and/or its underlying database) at points of interest. Then, `PROPOLIS` computes (offline) provenance expression for the given LTL formula with respect to the process specification. This expression is the compact result of evaluating the LTL formula with respect to *applying all possible combinations of specified scenarios (parameter values) to the process specification.* The provenance expression is passed to a module which allows for rapid exploration of scenarios as well as provenance specialization, using the provisioned expression while avoiding further costly access to the database or costly reevaluation of the LTL formula.

Experiments (Section 6). Finally, we have conducted an experimental study designed to examine the performance of the approach with respect to several measures. First, we have measured *provenance generation time* and *size of the obtained expression*, as a function of both the database size and the size of the FSM capturing the specification logical flow. We have studied these for various DDPs, both synthetic and based on real workflows. We show that the

(offline) expression generation time scales well with growing input size, and that the obtained expression size also grows in a reasonably moderate way. We further demonstrate that, once the expression is computed, *using it for exploring* scenario combinations can be done very efficiently. We have compared the time it takes to use the expression, with a simple baseline alternative approach. The alternative approach applies the hypothetical scenarios directly to the process specification and performs the analysis on the modified application. We have observed that the provenance-based approach significantly outperforms this alternative approach.

We review related work in Section 7 and conclude with directions for future work in Section 8.

2. DATA-DEPENDENT PROCESSES AND THEIR ANALYSIS

We next define our model for data-dependent processes (DDPs) and the temporal formalism used in its analysis. Provenance will be introduced in Section 3.

2.1 Data-dependent Processes

We consider a simple model for DDP specifications. The logical flow is specified by a state machine in which some transitions are governed by queries in a class $\mathcal{Q}$ over an underlying Database of schema $\mathcal{D}$. We will later explain how analysts are able to examine the effects of changes to the specification or database. We focus on guarding queries in the positive relational algebra with (possibly nested) aggregates and use SQL syntax for them.

DEFINITION 2.1. *A Data-Dependent Process (DDP) specification is a tuple* $(V, E, V_q, V_e, v_{init}, F_q, D)$ *such that* (V, E) *is a directed graph referred to as the DDP state machine,* $V_q, V_e \subseteq V$ *are subsets of nodes referred to as* query nodes *and* external effect nodes *respectively, such that* $V_q \bigcap V_e = \emptyset$ *and* $V_q \bigcup V_e = V$. *There is a distinguished "initial" node* $v_{init} \in V_q \bigcup V_e$. *Every node* $v_q \in V_q$ *has exactly two outgoing edges, and the end nodes of these edges are denoted* $true(v_q)$ *and* $false(v_q)$. $F_q : V_q \mapsto \mathcal{Q}$ *maps query nodes to queries deciding their transitioning. Finally,* D *is a database over the schema* $\mathcal{D}$.

The DDP model is based on the abstract FSM flow model; there are no restrictions on the nodes, or on the FSM topology, which in particular may include cycles. The intuition is that the DDP captures the logical application flow, as well as its data-dependency.

EXAMPLE 2.2. *Consider the (partial) process logic in Figure 1. Each node intuitively stands for a web page, and edges model links. The initial node is HomePage. Some transitioning is based on the user decision, such as the one from "Shopping Cart", depending on the user navigation choice (ignore for now the numbers annotating some transitions). Other transition choices depend on the* underlying database*: for instance the transition from "Cat." (standing for a page where the user chooses a category) to a "SubCat." page (where the user is presented a set of sub-categories) or to a "Product" page (listing relevant products) depends on availability of sub-categories, modeled as the truth value for the boolean query "*$Q_1 = 0$*" (checking for inexistence of a sub-category of available categories).* Q_1 *is given in Fig. 3, with respect to the underlying database whose fragment is given in Fig. 2 (ignore for now the Prov. column).*

A valid execution of a DDP follows query results for transitioning out of query nodes, while performing arbitrary choices for other nodes. We limit our attention to finite executions. To simplify definitions, we also assume that an execution terminates in a node from which there is no outgoing edge (*Exit* and *PayExit* in the running example).

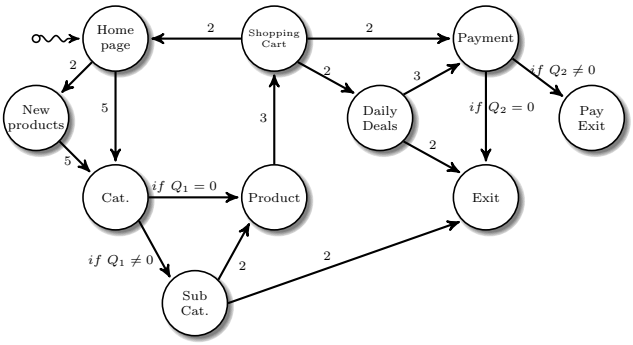


Figure 1: Data-Driven Process

DEFINITION 2.3 (EXECUTIONS). *An execution of a DDP* $(V, E, V_q, V_e, v_{init}, F_q, D)$ *is a finite path* $[v_1, v_2, ..., v_n]$ *in* (V, E) *s.t.* $v_1 = v_{init}$, v_n *has no outgoing edges, and for every* $v_i \in V_q$, *if the query* $F_q(v_i)$ *is satisfied by the database* D *then* $v_{i+1} = true(v_i)$ *and otherwise* $v_{i+1} = false(v_i)$.

AvailableCat

Cat.	Prov.
...	...
Cell Phones	d_1
Computers	d_2
Fashion	d_3
...	...

PaySys

Cat.	Prov.
...	...
PayPal	d_5
...	...

CategoryHierarchy

Cat	SubCat.	Prov.
...	...	...
Cell Phones	Smartphones	d_4
...	...	...

Figure 2: Underlying Database

EXAMPLE 2.4. *Reconsider the DDP specification in Example 2.2. Consider the path* $P = [Homepage, Cat, SubCat, Exit]$, *intuitively corresponding to a user making a choice of a category, then of a sub-category, then exiting without completing a purchase. Cat. is the only query node (i.e. Cat.* $\in V_q$*) in this execution, associated with the query* $Q_1 = 0$. *Since the current state of* D *does not satisfy* $Q_1 = 0$, *the node that follows Cat must be SubCat, so* P *is an execution but e.g.* $P' = [HomePage, Cat, Product, ...]$ *is not.*

Q_1 :

```
SELECT COUNT(*)
FROM CategoryHierarcy CH,
     AvailableCat AC
WHERE CH.Cat = AC.Cat
```

Q_2 :

```
SELECT COUNT (*)
FROM PaySys PS
```

Figure 3: SQL Queries

Note that a DDP may admit infinitely many (finite) executions, if its FSM includes cycles. We next consider LTL as a formalism that allows to specify properties of interest with respect to such executions.

2.2 Analysis formalism

We revisit the syntax of LTL formulas [29], as well as their semantics with respect to finite process executions [20] [1]. We

[1]The semantics of LTL is more commonly defined with respect to infinite executions; for provenance we are only interested in finite prefixes, as is also the case in [20].

follow common practice [20] of considering for finite runs only the next-free fragment of LTL (we may also account for the *next* operator with particular choice of semantics for the last state). Formulas are built up from a finite set of propositional variables P, the connectives $\neg$ and $\vee$, and the *modal operator* U (until). I.e. if $p \in P$ then p is an LTL formula, and for every two LTL formulas f_1,f_2, it holds that $f_1 \bigvee f_2$, $\neg f_1$, and $f_1 U f_2$ are LTL formulas.

The semantics of LTL formulas defines their satisfaction w.r.t. an *execution* with the assumption that each state can be tested for satisfaction of a variable in P. In our examples P corresponds to states names, and a state only satisfies the variable corresponding to its name.

DEFINITION 2.5. *[20] Let ϕ be an LTL formula over a set of propositional variables P. Let $e = v_0, v_1, ..., v_n$ be an execution and for $0 \leq i \leq n$ let $e^i = v_i, v_{i+1}, ...v_n$. We say that $e \models \phi$ if*

- *ϕ is a propositional variable p and v_0 satisfies p.*
- *$\phi = \neg\phi_1$ for some LTL formula ϕ_1 and $e \not\models \phi_1$.*
- *$\phi = \phi_1 \vee \phi_2$ for some LTL formulas ϕ_1 and ϕ_2 such that $e \models \phi_1$ or $e \models \phi_2$.*
- *$\phi = \phi_1 U \phi_2$ and there exists some $0 \leq i \leq n$ such that $e^i \models \phi_2$, and for all $0 \leq k < i$, $e^k \models \phi_1$.*

The basic modal operator U allows the definition of many derived operators such as "Finally" ($F\phi$), and "Before" ($\phi_1 B \phi_2$). $F\phi$ states that ϕ holds eventually, and it can be written as $true\ U\phi$. The formula $\phi_1 B \phi_2$ state that ϕ_1 holds before ϕ_2 and can be written as $\neg\phi_2 U \phi_1$.

EXAMPLE 2.6. *An analyst of our running example DDP may be interested in various execution properties, such as whether the execution involves "A user exiting without viewing the daily deals". This is captured by the LTL formula* (Exit $\vee$ PayExit) B DailyDeals. *Other properties that may be of interest to the analyst are "A user views product subcategories for some category" or "A user views proposed daily deals, and later proceeds to the payment page". These properties can be captured by the LTL formulas* F SubCat.*, and* F (DailyDeals $\wedge$ F Payment) *respectively.*

Note that an LTL formula expresses a property of a given execution, while in general we are interested in evaluating the formula with respect to the (possibly infinitely many) possible executions of a given DDP. We next define the (boolean) semantics of such evaluation.

DEFINITION 2.7. *Given a DDP s and a LTL formula ϕ, we say that $s \models \phi$ if there exists an execution e of s such that $e \models \phi$* [2].

The above definition is essentially restricted to asking for the existence of a path satisfying the formula, and does not account for neither data in meta-domains nor hypothetical reasoning. To this end we present a provenance model. The boolean semantics will serve as a yardstick in our development: we will generalize it.

[2]A different common semantics asks that *every* execution satisfies ϕ; since LTL is closed under negation, moving between the two semantics is straightforward and we find this one more natural for the provenance settings.

3. PROVENANCE MODEL

We present in this section a semiring-based provenance model for the LTL-based analysis of DDPs. The section is organized as follows. We start by recalling the notion of semirings and use them to introduce the notion of annotated DDPs (Section 3.1). We then recall the way in which provenance propagates through database queries (Section 3.2). We then explain the need for a novel structure that allows capturing provenance for DDP executions. We introduce the structure as well as the provenance definitions (Section 3.3). Finally we show (Section 3.4) how to refine the structure by introducing a congruence relation that allows simplifying the obtained expressions.

3.1 Semiring and Annotated DDPs

We start by recalling the notion of a semiring and related notions (see [23] for more details), then explain how to use semirings to annotate DDPs.

A *commutative monoid* is an algebraic structure $(M, +_M, 0_M)$ where $+_M$ is an associative and commutative binary operation and 0_M is an identity for $+_M$. A monoid homomorphism is a mapping $h : M \to M'$ where M, M' are monoids, and $h(0_M) = 0_{M'}$,$h(a +_M b) = h(a) +_{M'} h(b)$. We will consider database operations on relations whose tuples are annotated with elements from *commutative semirings*. These are structures $(K, +_K, \cdot_K, 0_K, 1_K)$ where $(K, +_K, 0_K)$ and $(K, \cdot_K, 1_K)$ are commutative monoids, $\cdot_K$ is distributive over $+_K$, and $a \cdot_K 0_K = 0 \cdot_K a = 0_K$. A semiring homomorphism is a mapping $h : K \to K'$ where K, K' are semirings, and $h(0_K) = 0_{K'}, h(1_K) = 1_{K'}$,$h(a +_K b) = h(a) +_{K'} h(b)$, $h(a \cdot_K b) = h(a) \cdot_{K'} h(b)$. Examples of particular interest to us include the boolean semiring $(\{true, false\}, \vee, \wedge, false, true)$ and the *tropical* semiring $(\mathbb{N}^\infty, min, +, \infty, 0)$ (also termed a *cost* semiring) where $\mathbb{N}^\infty$ includes all natural numbers as well as ∞. Given a set X of *provenance tokens* which correspond to "atomic" provenance information, e.g., tuple identifiers, the semiring of *polynomials* with natural coefficients $(\mathbb{N}[X], +, \cdot, 0, 1)$ was shown in [23] to most generally capture provenance for positive relational queries.

We are now ready to define the notion of a Provenance-Aware DDP (PADDP for short). We propose to use *two* semirings for annotations, accounting for the inherently different types of transitioning. A first semiring will be used to annotate the database tuples, following [23]; a second semiring will be used for annotation of external effects.

DEFINITION 3.1. *A Provenance-Aware DDP (PADDP) is a tuple $(S, K_{ext}, K_{data}, A_{ext}, A_{data})$ where S is a DDP, K_{ext}, K_{data} are commutative semirings, A_{ext} maps transitions out of external effects nodes of S to elements of K_{ext}, and A_{data} maps tuples of the underlying database of S to elements of K_{data}.*

For any two commutative semirings K, K' we will use the term (K,K')-PADDP for a PADDP whose annotations for external effect are elements of K and annotations for data are elements of K'.

We have not defined yet how these annotations propagate through executions, but we can already exemplify their use based on the correspondence of the semiring + and $\cdot$ operations with *alternative and joint use* (of data / transitions).

EXAMPLE 3.2. *Re-consider the running example and suppose that the analyst is interested in questions of the flavor "What is the minimum user effort (quantities associated*

with e.g. following a link, filling in a text box etc.) required for a user to view the daily deals and later proceed to the Payment page. We have already shown how this constraint on executions can be expressed in LTL, but we need to also capture user effort. This can be done by choosing K_{ext} to be the tropical semiring *(identifying "cost" here with user effort), and using A_{ext} to map transitions to natural numbers corresponding to user effort associated with them (see weights next to transitions in Figure 1). Since multiplication in the tropical semiring corresponds to natural numbers addition, weights of joint choices along an execution are summed up; and since addition corresponds to* min*, provenance for multiple executions (e.g. all those satisfying the constraint that daily deals are viewed) captures their smallest weight. We will show concrete examples for the propagation of such weights in Examples 3.7 and 3.11.*

Furthermore, the analyst may be interested in performing the analysis assuming the database will change*, e.g. some products or categories are no longer available. This can be captured by associating provenance with the database tuples. Specifically, we can use $K_{data} = \mathbb{N}[D]$, the semiring of polynomials over the set of provenance tokens $D = \{d_1, ..., d_5\}$. The annotation function A_{data} is given as the Prov. column in Figure 2. Intuitively, D may be considered as indeterminates, and hypothetical scenarios will be modeled via truth assignments (see Example 4.5).*

3.2 Provenance for Database Queries

We briefly recap the provenance model established in [23, 4], and refer the reader to these references for further details.

Positive Relational Algebra on K-Databases. Intuitively, a K-database is a database whose tuples are associated with annotations taken from the semiring K. Then, the (positive) *K-relational algebra* defined in [23] corresponds to a semantics on K-databases for the usual operations of the relational algebra, based on the intuitive correspondence between + operation and *alternative use* of data (such as in union and projection), and the $\cdot$ operation as the *joint use* of data (as in cartesian product and join), the use of 1 for annotation of data that is always available (we do not track its provenance), and 0 standing for data that is absent.

Queries involving aggregates. In [4] we have observed that the semiring framework of [23] cannot adequately capture aggregate queries. To solve the problem we have further generalized K-relations by extending their data domain with *aggregated values*. In this extended framework, relations have provenance also as part of their values, rather than just in the tuple annotations. Such a value is a *formal sum* $\sum_i t_i \otimes v_i$, where v_i is the value of the aggregated attribute in the i^{th} tuple, while t_i is the provenance of that tuple. We can think of $\otimes$ as an operation that "pairs" values (from a monoid M) with provenance annotations. In [4] the framework was also used to define provenance for *nested aggregates* and *negation* by introducing equation elements. Intuitively an equation such as $[(d_1 \cdot d_2) \otimes m = 0]$ is kept as an abstract "token" and can be used in conjunction with other semiring elements. Given concrete values for d_1, d_2 and m one may test the truth value of the equality and "replace" the equation by the truth value [3]. A precise algebraic treatment of aggregated values and the equivalence laws that govern them is based on semimodules and is described in [4].

Now that we have defined provenance for database queries, we can associate provenance with *individual transitions* of the DDP. We have already mentioned that transitioning for external effect nodes is associated with elements of K_{ext}; and we can now associate transitioning out of query nodes with the query provenance (in K_{data}). Our next goal is to define provenance for a possibly infinite set of execution paths.

3.3 Provenance for PADDP Executions

We next introduce a provenance structure that can accommodate provenance for PADDP executions. We start by presenting the construction in a fairly general way and will then show how to use it in our context. Let K and L be two semirings. We start our construction with a simple step, introducing the set of *pairs* over items of K and L, denoted $K \times L$. Following common practice we denote its elements $k \otimes l$ as well as $\langle k, l \rangle$ invariably. The intuition is that a single pair $k \otimes l$ will capture the provenance of an entire execution: k will capture the "external provenance" of the execution and l will capture its "data provenance". A significant difficulty that is addressed lies in correctly managing provenance for possibly *infinitely many* alternatively executions, accounting for operations on these pairs.

Next, we consider the set $Bag(K \times L)$ of *possibly infinite bags* of such pairs. There are two useful ways of working with bags. One is to consider them as $\mathbb{N}^\infty$-valued functions. The other is to observe that bags, with bag union and the empty bag, form a commutative monoid whose elements are uniquely representable as infinite sums of singleton bags. In this second perspective, if we abuse notation and denote singleton bags by the unique element they contain we can write each bag of pairs from $K \times L$ as

$$\sum_{i \in I} k_i \otimes l_i$$

where i ranges over an infinite set of indices I. Moreover, we can always rename the indices without loss of generality such that when we work with several bags, the sets of indices used in their sum representations are pairwise disjoint.

It will be convenient to already denote bag union by $+_{K \otimes L}$ and the empty bag by $0_{K \otimes L}$. That is,

$$(\sum_I k_i \otimes l_i) +_{K \otimes L} (\sum_J k_j \otimes l_j) = \sum_{I \cup J} k_h \otimes l_h$$

$$0_{K \otimes L} = \sum_{\emptyset} k_i \otimes l_i$$

Recalling the intuitive correspondence between summation and alternative use, a bag (sum) of pairs corresponds to alternative paths. Now we define

$$(\sum_I k_i \otimes l_i) \cdot_{K \otimes L} (\sum_J k_j \otimes l_j) = \sum_{I \times J} (k_i \cdot_K k_j) \otimes (l_i \cdot_L l_j)$$

This is again consistent with the intuition of multiplication as joint use and summation as alternatives: following one of the alternatives of the first bag combined with one of the alternatives of the second bag, corresponds exactly to all alternatives obtained by such combinations

[3]The obtained semiring is denoted in [4] by K^M. For simplicity we will abuse notation here and just use K

We have defined a mathematical structure to capture bags of pairs. For this structure to be consistent with our intuition of summation capturing alternatives and product capturing joint use, they should follow the axioms of commutative semiring (e.g. associativity, distributivity etc.). Indeed, we may show that this is the case.

PROPOSITION 3.3. *$(Bag(K\times L), +_{K\otimes L}, \cdot_{K\otimes L}, 0_{K\otimes L}, 1_K \otimes 1_L)$ is a commutative semiring.*

Overloading notation we will use $Bag(K \times L)$ to denote $(Bag(K\times L), +_{K\otimes L}, \cdot_{K\otimes L}, 0_{K\otimes L}, 1_K \otimes 1_L)$. The following proposition shows that the obtained structure is closed (see [36] for a definition). Intuitively, infinite sums in the structure interact in the "expected" way with semiring operations. In particular, this means that they preserve the intuition of correspondence with joint and alternative use in executions.

PROPOSITION 3.4. *$Bag(K \times L)$ is a closed semiring.*

We are now ready to define provenance for executions of PADDPs. Provenance for a transition outgoing an external choice node is annotated with a singleton bag $\{\langle k, 1_{K_{data}}\rangle\}$ where $k \in K_{ext}$ is the provenance associated with this transition according to the PADDP specification; and $1_{K_{data}}$ is the neutral value with respect to multiplication in K_{data}. Intuitively since there is no effect with respect to data provenance. Similarly provenance for a transition out of a query node is defined as $\{\langle 1_{K_{ext}}, k'\rangle\}$ where $k' \in K_{data}$ is the annotation obtained for the corresponding query with respect to the underlying annotated database.

Given a transition t of a PADDP we use $Prov(t)$ to denote the transition provenance according to the above. The provenance of an execution is then the multiplication of provenance expressions associated with its transitions.

DEFINITION 3.5. *Given a PADDP and an execution $e = (v_0, v_1, ..., v_n)$, the provenance of e, denoted $Prov(e) \in Bag(K_{ext}\times K_{data})$, is defined as $\prod_{(v_i,v_{i+1})\in e} Prov((v_i, v_{i+1}))$.*

Note that the provenance of an execution involves multiplication of bags in $Bag(K_{ext}\times K_{data})$. This allows to "mix" annotations of K_{data} and annotations in K_{ext} in the same expression. To simplify notations, we will identify $\langle k, k'\rangle$ with the singleton bag $\{\langle k, k'\rangle\}$.

EXAMPLE 3.6. *Reconsider the PADDP of Example 3.2, the provenance of [Home page, Cat., Sub Cat., Exit] is* $\langle 5, 1\rangle \cdot \langle 0, [d_1 \cdot d_4 \neq 0]\rangle \cdot \langle 2, 1\rangle$

Each pair element in this multiplication corresponds to a single transition, and they are multiplied since they are used together in an execution.

For example, the item $\langle 5, 1\rangle$ stems from the (external effect) transition from HomePage to Cat and is shorthand to the singleton bag containing of a pair whose first element is the natural number 5 in the tropical semiring (signaling user cost of 5), and its second element 1 is the neutral with respect to multiplication in $\mathbb{N}[D]$.

The item $\langle 0, [d_1 \cdot d_4 \neq 0]\rangle$ originates in the transition from Cat. to Sub Cat. depending on the underlying database; note that the natural number 0 is the neutral with respect to multiplication of the tropical semiring. $[d_1 \cdot d_4 \neq 0]$ is in fact shorthand to $[[(d_1 \cdot d_4) \otimes 1 = 0] = 0]$ which is the negation of the provenance for $Q_1 = 0$, itself computed through the framework for database queries described in Section 3.2. Intuitively, it means that both tuples annotated with d_1 and d_4 need to be present for the query to be satisfied.

Finally, we can use the definition of bag multiplication above to simplify the expression by performing "point-wise multiplication", to obtain ($\cdot_T$ and $\cdot_{\mathbb{N}[D]}$ stand for multiplications in the tropical and $\mathbb{N}[D]$ semirings, resp.; for brevity the subscript of operation is omitted in the sequel where clear from context).

$\langle 5 \cdot_T 0 \cdot_T 2, 1 \cdot_{\mathbb{N}[D]} [d_1 \cdot d_4 \neq 0] \cdot_{\mathbb{N}[D]} 1\rangle \equiv \langle 7, [d_1 \cdot d_4 \neq 0]\rangle$

Note that multiplication in the tropical semiring corresponding to natural number additions. The accumulated cost ("user effort") for this path is 7 and the accumulated condition with respect to the database is $[d_1 \cdot d_4 \neq 0]$.

We next define provenance for an LTL formula with respect to a PADDP, as the (possibly infinite) sum of provenances of executions conforming to the formula. Let $exec(S)$ denote the (possibly infinite) set of (finite) executions of a PADDP S. We define:

DEFINITION 3.7. *Given a PADDP S and an LTL formula f, we define the result of evaluating f on S (denoted $f(S)$) as $\sum_{\{e\in Paths(S)|e\models f\}} prov(e)$.*

Definition 3.7 does not give an explicit way of representing the infinite sums. To this end, we introduce the *Kleene star* operation $a^* = 1+a+a^2+...$, and note that it is well-defined in closed semirings.

EXAMPLE 3.8. *In the tropical semiring we get $a^* = 0$ (the tropical 1, i.e. the natural number 0). For the Boolean semiring, we will get $a^* = true \vee a \vee a... = true$.*

We then say that a starred expression is an expression involving the star operation. The following proposition indicates that the proposed structure may facilitate provenance for analysis results.

PROPOSITION 3.9. *For any PADDP S and LTL formula f, $f(S)$ may be represented as a finite starred expression.*

We will give an algorithm to compute this starred expression in Section 4, but we already note that intuitively, the obtained starred expression corresponds to a regular expression over the annotation pairs, capturing exactly those paths that satisfy the LTL formula. Based on this intuition we exemplify the obtained expressions.

EXAMPLE 3.10. *Reconsider the LTL formula* F (Daily-Deals $\wedge$ F Payment) *and the running example PADDP. Intuitively, to represent alternative paths (alternative ways of "realizing" the LTL property) we use* sum of pairs where each pair captures the provenance of a single path. *Also, using the introduced axioms, we may in fact generate sub-expressions for multiple partial executions, and then combine them. For instance, the two simple partial executions reaching Cat yield a joint provenance which is the sum $\langle 5, 1\rangle + \langle 7, 1\rangle$. We may then continue constructing expressions for sub-executions, and eventually we can obtain:*

$\Big(\big(\langle 5, 1\rangle + \langle 7, 1\rangle\big) \cdot \big(\langle 2, 1\rangle \cdot \langle 0, [d_1 \cdot d_4 \neq 0]\rangle + \langle 0, [d_1 \cdot d_4 = 0]\rangle\big) \cdot \langle 3, 1\rangle \cdot \langle 2, 1\rangle\Big)^* \cdot \big(\langle 5, 1\rangle + \langle 7, 1\rangle\big) \cdot \big(\langle 2, 1\rangle \cdot \langle 0, [d_1 \cdot d_4 \neq 0]\rangle + \langle 0, [d_1 \cdot d_4 = 0]\rangle\big) \cdot \langle 3, 1\rangle \cdot \langle 2, 1\rangle \cdot \langle 3, 1\rangle \cdot \big(\langle 0, [d_5 \neq$

$0]\rangle + \langle 0, [d_5 = 0]\rangle\big)$

The obtained expression is quite long but we can already simplify it using the axioms of our structure. In particular, we have seen before that we can simplify multiplication expression, to get e.g. $\langle 3,1\rangle \cdot \langle 2,1\rangle \cdot \langle 3,1\rangle = \langle 8,1\rangle$. *Further simplifications lead to:*

$\Big(\big(\langle 5,1\rangle + \langle 7,1\rangle\big) \cdot \big(\langle 7, [d_1 \cdot d_4 \neq 0]\rangle + \langle 5, [d_1 \cdot d_4 = 0]\rangle\big)\Big)^* \cdot$
$\big(\langle 5,1\rangle + \langle 7,1\rangle\big) \cdot \big(\langle 10, [d_1 \cdot d_4 \neq 0]\rangle + \langle 8, [d_1 \cdot d_4 = 0]\rangle\big) \cdot$
$\big(\langle 0, [d_5 \neq 0]\rangle + \langle 0, [d_5 = 0]\rangle\big)$

In this expression, every pair represents "joint" provenance terms in the two domains (tropical and $\mathbb{N}[D]$*). For instance,* $\langle 5,1\rangle$ *intuitively means a cost of* 5 *and no dependency on data ("1" is the neutral element of* $\mathbb{N}[D]$*). A sum of such pairs reflects alternative paths, e.g. the sub-expression* $(\langle 5,1\rangle + \langle 7,1\rangle)$ *corresponds to the two options of following a sub-path with cost* 5 *or following one with cost* 7 *(with no dependency on the data). A product of such sub-terms corresponds to joint use (i.e. in conjunction with taking either of these transitions, we also continue the execution). Kleene star is applied to provenance of sub-executions appearing in a loop. The expression that we get is still quite complex but we will show later (Example 3.11), that by introducing congruence axioms, we can further simplify it.*

3.4 Introducing a Congruence

We have defined provenance for LTL formula evaluated over PADDPs, but observed that their representation may become quite complex. There are certain equivalence axioms that are "natural" in this setting. For instance if the same data provenance is used repeatedly in multiple execution paths, one expects to be able to write an equivalent expression where it appears only once.

To allow for simplifications, we need to identify elements of $Bag(K \times L)$ with other, "simpler" elements. This is done via the introduction of a congruence relation. Since we are dealing with possibly infinite bags, we need to consider *inifinitary congruences*, namely one that is preserved also by (in addition to multiplication and finite summation) infinite summation. Next, let $\sim$ be the smallest inifinitary congruence on $Bag(K \times L)$ with respect to $+_{K\otimes L}$ and $\cdot_{K\otimes L}$ that contains (for all k, k', l, l'):

$$\begin{aligned}
(k +_K k') \otimes l &\sim k \otimes l +_{K\otimes L} k' \otimes l \\
0_K \otimes l &\sim 0_{K\otimes L} \\
k \otimes (l +_L l') &\sim k \otimes l +_{K\otimes L} k \otimes l' \\
k \otimes 0_L &\sim 0_{K\otimes L}
\end{aligned}$$

We denote by $K \otimes L$ the set of congruence classes of bags of pairs modulo $\sim$ and by $1_{K\otimes L}$ the congruence class of $1_K \otimes 1_L$. As usual when we take the quotient by a congruence the result, $(K \otimes L, +_{K\otimes L}, \cdot_{K\otimes L}, 0_{K\otimes L}, 1_{K\otimes L})$, which we will also denote $K \otimes L$, is a commutative semiring.

When we define provenance for LTL queries in $K \otimes L$ all constructions and results go through, and in addition we can perform significant expression simplifications.

Example 3.11. *Reconsider the provenance expression obtained in Example 3.10. Note that the introduced equivalence axioms allow for simplifications that were not possible so far. For instance, we have the sub-expression* $\langle 5,1\rangle + \langle 7,1\rangle$*, intuitively corresponding to two sub-executions that involve the same dependency on data (in this case, no dependency) with different costs. Using the congruence relation, this expression is equivalent to* $\langle 5+7, 1\rangle = \langle 5, 1\rangle$. *Intuitively we have "factored out" the common dependency on data. Via repeated applications of the congruence relation we may perform further partial computations, and obtain:*

$\Big(\langle 5+7,1\rangle \cdot \big(\langle 7, [d_1 \cdot d_4 \neq 0]\rangle + \langle 5, [d_1 \cdot d_4 = 0]\rangle\big)\Big)^* \cdot \langle 5+7,1\rangle \cdot$
$\big(\langle 10, [d_1 \cdot d_4 \neq 0]\rangle + \langle 8, [d_1 \cdot d_4 = 0]\rangle\big) \cdot \big(\langle 0, [d_5 \neq 0]\rangle +$
$\langle 0, [d_5 = 0]\rangle\big)$
$= \big(\langle 12, [d_1 \cdot d_4 \neq 0]\rangle + \langle 10, [d_1 \cdot d_4 = 0]\rangle\big)^* \cdot \big(\langle 15, [d_1 \cdot d_4 \neq 0]\rangle + \langle 13, [d_1 \cdot d_4 = 0]\rangle\big) \cdot \big(\langle 0, [d_5 \neq 0] + [d_5 = 0]\rangle\big)$

In Section 4 we show further simplifications are possible after concrete truth values are assigned to the d's.

An important property of the construction thus far was that the obtained semiring was closed, which allowed the definition of infinite sums. This still holds:

Proposition 3.12. *For every two semirings* K, L*, it holds that* $K \otimes L$ *is a closed semiring.*

We next note that the additional identities that we have forced by taking the congruence provide a key "universality" property of this semiring. Define $\iota_K : K \to K \otimes L$ where $\iota_K(k)$ is the congruence class of $k \otimes 1_L$ and $\iota_L : L \to K \otimes L$ where $\iota_L(l)$ is the congruence class of $1_K \otimes l$.

Proposition 3.13. ι_K *and* ι_L *are semiring homomorphisms; for any other commutative semiring* H *and any two semiring homomorphisms* $f : K \to H$ *and* $g : L \to H$ *there exists a unique semiring homomorphism denoted* $f \otimes g : K \otimes L \to H$ *such that* $f = (f \otimes g) \circ \iota_K$ *and* $g = (f \otimes g) \circ \iota_L$.

This universality property fits well with the intuition behind tracking annotations. We use $K \otimes L$ to track both K-annotations and L-annotations while at the same time each of K and L are "embedded" into $K \otimes L$ by the ι's so in $K \otimes L$ we can track K-annotations while L-annotations are kept neutral, and vice-versa.

Finally, a sanity check is that the definition is consistent with the LTL semantics for DDPs without annotation, namely that it in fact *faithfully extends* it.

Proposition 3.14. *For any LTL formula* f *and* $(\mathbb{B}, \mathbb{B})$*-PADDP* S*, it holds that* $f(S) \equiv \{(true, true)\}$ *if and only if* $S' \models f$ *where* S' *is a DDP obtained from* S *by deleting all tuples and transitions annotated by* $false$*, and keeping the rest with no annotation.*

4. COMPUTING AND USING PROVENANCE

We have defined in the previous section a model for provenance for DDPs and for LTL queries on its execution. Given the model, the two main challenges that remain are (1) computation of a finite provenance representation and (2) establishing means for using the obtained expression. In this section we consider these two challenges.

4.1 Computing Provenance for LTL

We start by proposing an algorithm for computing the provenance of an LTL expression with respect to a PADDP.

Proposition 4.1. *For any LTL formula* ϕ *and a PADDP* S *we can compute a description of* $\phi(S)$ *in time polynomial*

in the size of the underlying database and exponential in the size of the state machine of S [4].

PROOF. (sketch) The high-level flow of an algorithm that generates provenance for an LTL formula with respect to a PADDP is given in Algorithm 1. The first step is to compute provenance expressions for the queries associated with query nodes, as described in Section 3.2, and the obtained annotated structure is named S'. The LTL formula is compiled (Line 2) into an FSM consistent with its finite semantics (see [20]), and *intersected* with S' (Line 3). The intersection is performed in the standard way, while maintaining annotations of S'' consistent with those of S': namely, a transition in the intersection automata S'' that has origin (u, ψ) and destination (v, ψ') will be annotated by the annotation in S' of (u, v). Note that S'' is a PADDP with some states designated as accepting. The last step (line 4) is to transform S'' into a starred expression, which is an element of the tensor product semiring. This is done by applying Kleene's algorithm (see e.g. [36]) (which is a generalization of the standard translation of FSMs to regular expressions), interpreting the addition, multiplication and Kleene star operation as their counterparts in the tensor product. Following the correctness of Kleene's algorithm [36], the computed expression is equivalent to $\phi(S)$, capturing exactly provenance of all executions of S satisfying ϕ.

Complexity and Output size. The generation of provenance expressions for database queries was shown to be in polynomial time with respect to the database size. The translation to starred expression (step 4), however, may in the worst case incur an (unavoidable [24]) exponential blow up (both of the expression size, and execution time of the algorithm) in the size of the state machine. □

Algorithm 1 Algorithm for Expression Genretaion

Input PADDP specification S; LTL formula ϕ
Output Provenance expression $\phi(S)$
1: $S' := QueriesToProvenance(S)$
2: $QueryAutomaton := TransformToFSM(\phi)$
3: $S'' := Intersect(S', QueryAutomaton)$
4: $exp := TranslateToStarredExpression(S'')$
5: **return** exp

Discussion. In the worst case, the provenance size may be exponential in the state machine size (but not in the database size). This is unavoidable in general due to lower bounds on translating FSMs to regular expressions [24]. However, we note that in many practical cases both the generation time and the size of obtained expressions may be feasibly small, as follows.

First, with regards to dependency on the database size, we note that the size is polynomial in the number of tuple annotations (which is bounded by the database size but in practice may be much smaller, since tuples may be annotated by neutral semiring values). Furthermore some input tuples may contribute nothing to the result of guarding queries (as is e.g. the case with high query selectivity): this will further reduce the size of the expression. This was observed in the context of database querying in previous work (see e.g. [22]) and is further validated in our experiments.

[4]We follow common practice of analyzing data complexity: ϕ and guarding queries are considered of constant size.

Second, the part of the algorithm leading to the exponential blow-up w.r.t. FSM size is line 4, which involves the translation of the (intersected) DDP to a starred expression. In the worst case this blow-up is unavoidable, as a corollary of a similar bound on translating FSMs to regular expressions [24]. However, this may be addressed as follows. First, for restricted yet useful FSM structures, the translation algorithm yields in practice small-size expressions (see for example the experiments in section 6). We also note that the time complexity of the algorithm is polynomial in the size of expression that it generates. Moreover, for many restricted, yet expressive, classes of FSMs we may obtain polynomial (and even linear) complexity bounds by leveraging and adapting dedicated algorithms, and "plugging" them in as implementation of line 4 of the algorithm. Such classes include e.g. *SP-Automata* [34], *UDR-Automata* [33] and Thompson graphs [19]. Even when the workflow in hand follows a more complex structure, there are strong practical optimizations that may be effectively adapted and employed (see e.g. [25]). Last, we note that we expect that in real-life DDPs the FSM (which reflects the logical application flow and is often created manually) size will be of small size comparing to the underlying database size.

4.2 Using Provenance

An important principle underlying the semiring-based provenance framework is that one can compute an "abstract" provenance representation and then "specialize" it in any domain. This "specialization" is formalized through the use of *semiring homomorphism*. To allow for a similar use of provenance in our setting, we extend the notion of homomorphism to the tensor product structure, and study properties of the construction. A fundamental new challenge lies in the mapping from elements of the tensor (which are essentially bags of pairs), to single semiring elements. We first show how to "shift" between meta-domains. The idea is that two mappings from the individual semirings may be combined in a single homomorphism over the tensor.

PROPOSITION 4.2. *Let K_1, K_2, K_3, K_4 be 4 semirings, and let $h_1 : K_1 \mapsto K_3$, $h_2 : K_2 \mapsto K_4$ be semiring homomorphisms. Let $h(\{k_1 \otimes k_2\}) = \{h_1(k_1) \otimes h_2(k_2)\}$ and extend h to a full mapping by defining $h(s_1 + s_2) = h(s_1) + h(s_2)$ and $h(s_1 \cdot s_2) = h(s_1) \cdot h(s_2)$. Then $h : K_1 \otimes K_2 \mapsto K_3 \otimes K_4$ is a semiring homomorphism.*

We use $h_1 \mid h_2$ to denote the homomorphism h obtained, according to the above construction, from h_1, h_2.

We can now extend the notion of semiring homomorphism to homomorphisms on PADDPs in a natural way:

DEFINITION 4.3. *Let K_1, K_2, K_3, K_4 be 4 semirings, let $h_1 : K_1 \mapsto K_3$, $h_2 : K_2 \mapsto K_4$ be semiring homomorphisms and let S be a (K_1, K_2)-PADDP. We use $(h_1 \mid h_2)(S)$ to denote the (K_3, K_4)-PADDP S' obtained from S by replacing every annotation $k_1 \in K_1$ by $h_1(k_1)$ and every annotation $k_2 \in K_2$ by $h_2(k_2)$.*

Crucially, we may show that provenance propagation *commutes with homomorphisms*. This will allow to support applications such as deletion propagation, as exemplified next.

PROPOSITION 4.4. *For every LTL formula ϕ, a (K_1, K_2)-PADDP S and semiring homomorphisms $h_1 : K_1 \mapsto K_3$, $h_2 : K_2 \mapsto K_4$, it holds that $(h_1 \mid h_2)(\phi(s)) \equiv \phi((h_1 \mid h_2)s)$.*

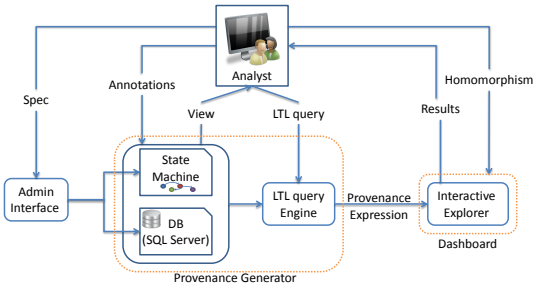


Figure 4: System Architecture

EXAMPLE 4.5. *We next show how to find the minimal cost of realizing the LTL query, but this time* under the assumption that the Cell Phone category is no longer available. *This corresponds to deleting the tuple annotated by* d_1 *(while other tuples are kept), and re-performing the analysis. But Proposition 4.4 suggests a much more efficient alternative: given the computed provenance expression of Example 3.11, we can use the homomorphism* $h_2 : \mathbb{N}[D] \mapsto \mathbb{B}$*, where* $h_2(d_1) = false$*,* $h_2(d_4) = true$ *and* $h_2(d_5) = true$*. We use the identity function as* h_1 *(assuming the user effort quantification stays intact), and obtain:*

$$(h_1 \mid h_2)\Big(\big(\langle 12, [d_1 \cdot d_4 \neq 0]\rangle + \langle 10, [d_1 \cdot d_4 = 0]\rangle\big)^* \cdot \big(\langle 15, [d_1 \cdot d_4 \neq 0]\rangle + \langle 13, [d_1 \cdot d_4 = 0]\rangle\big) \cdot \big(\langle 0, [d_5 \neq 0] + [d_5 = 0]\rangle\big)\Big) = \big(\langle 12, false\rangle + \langle 10, true\rangle\big)^* \cdot \big(\langle 15, false\rangle + \langle 13, true\rangle\big) \cdot \langle 0, true\rangle$$

and via more simplifications we obtain: $\langle 13, true\rangle$

Indeed, the minimal effort required to reach the goal given the deletions is 13*.*

Finally, it may be desirable that the provenance result reflects a single value (e.g. a cost, or a truth value) rather than a bag of pairs. This requires one last step, as follows.

PROPOSITION 4.6. *Let* K *be a positive semiring and let* $h : K \otimes K \mapsto K$ *be the mapping defined by* $h(k_1 \otimes k_2) = k_1 \cdot k_2$*, and extended to a full mapping by defining* $h(s_1 + s_2) = h(s_1) + h(s_2)$ *and* $h(s_1 \cdot s_2) = h(s_1) \cdot h(s_2)$*. Then* h *is a semiring homomorphism.*

Now provenance computation can be done in $K \otimes L$ for arbitrary positive K and L; then, to get an item (not a pair) in K (symmetrically L) we can first apply the homomorphism $h : K \otimes L \mapsto K \otimes K$ that combines (via prop. 4.2) the two mappings $h_1 : K \mapsto K$ defined by $h_1(k) = k$ and $h_2 : L \mapsto K$ defined by $h_2(l) = 0_K$ if $l = 0$ and $h_2(l) = 1_K$ otherwise. By Prop. 4.4 we get a correct provenance expression in $K \otimes K$. This last expression can then be mapped to an expression in K using the homomorphism in Prop. 4.6.

EXAMPLE 4.7. *Reconsider the obtained provenance expression* $\langle 13, true\rangle$*. We use the identity homomorphism* h_1 *and the homomorphism* h_2 *from boolean to tropical mapping true to the tropical* 1 *element and* $false$ *to the tropical* 0 *element. Composing the mapping in Prop. 4.6 with* $h_1 \mid h_2$*, and applying the result to* $\langle 13, true\rangle$*, we finally get* 13 *as final answer under the particular hypothetical scenario.*

5. PROTOTYPE IMPLEMENTATION

We have implemented our provenance framework in the context of `PROPOLIS` (PROvisioned Process Analysis) [13]. The system architecture is shown in Fig. 4 and we next describe its main components.

Provenance Generator. Interacting with the *Provenance Generator* module, analysts can design (through a dedicated GUI) a PADDP and an LTL query. The annotations may be taken from a variety of semirings (for trust, cost, probabilities, access control etc.). One use case involves the analyst deciding on a set of state machine transitions and database tuples that she wishes to *parameterize*, i.e. identify with abstract variables and defer the assignment of concrete values. This corresponds to using $\mathbb{N}[X]$ and $\mathbb{N}[D]$ as semirings for annotation, where X and D are indeterminates ("parameters") associated by the analyst with transitions and tuples. The output is a single memory-resident provenance expression, computed (offline) based on Algorithm 1.

Dashboard. Once the provenance expression has been computed, the analyst can interactively explore the effect of *hypothetical scenarios* on the LTL analysis result. The scenarios are expressed through the assignment of different *values*, from semirings of the analyst's choice (e.g. boolean, costs etc.) to the different parameters. This defines a *homomorphism* of the analyst choice to a chosen structure. A simple example involves deciding on a subset of tuples / transitions that are (hypothetically) deleted (see Examples 4.5 and 4.7). The analyst is presented with the query result for the specified scenario, and can repeatedly and interactively change the scenarios to explore the effect on analysis results.

6. EXPERIMENTAL EVALUATION

We have conducted experiments whose main goals were examining (1) the scalability of the approach in terms of the generated provenance size and generation time, and (2) the extent of usefulness of the approach, namely the time it takes to specialize the provenance expression for applications such as those described in this paper.

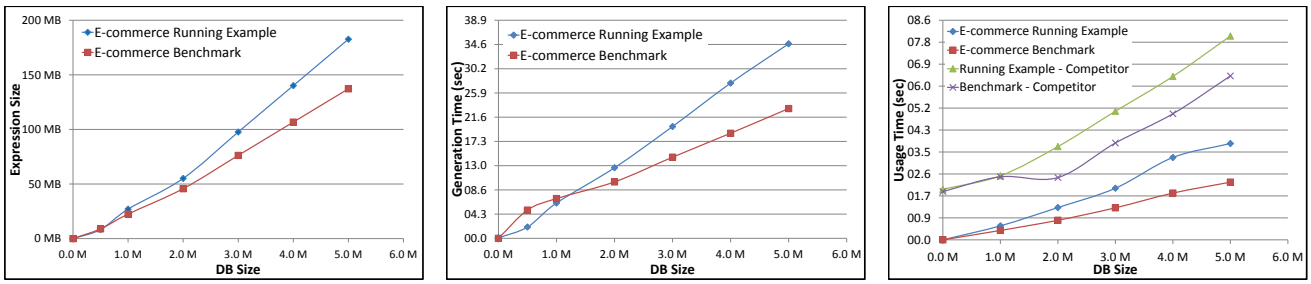
All experiments were executed on Windows 7, 64-bit, with 4GB of RAM and Intel Core Duo i5 3.10 GHz processor.

6.1 Evaluation Benchmark

We have developed a dedicated benchmark that involves both synthetic and real data as follows.

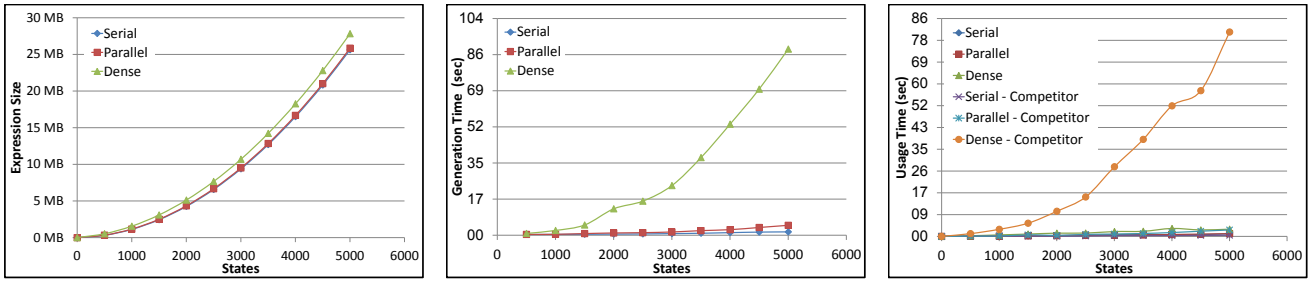
E-commerce (Synthetic dataset). We used two different DDPs for E-commerce process specifications to examine the performance (provenance size, and generation and usage time) as a function of the *underlying database size*. The first dataset uses the fixed topology of the state machine used in our running example (Figure 1), which demonstrates all features of the model, including an FSM with cycles, queries on a database and external choices. The second DDP is partially based on the proposed benchmark for e-commerce applications, described in [21]. The underlying database in both cases was populated with synthetically and randomly generated data, of growing size, *up to 5M tuples* (to examine scalability w.r.t the database size).

Arctic Stations (Real dataset). The second dataset is based on the "Arctic stations" data, used as a benchmark also in [3] (albeit in a different context, of tracking provenance of actual running executions). This benchmark includes a variety of processes that model the operation of meteorological stations in the Arctic. Their flows are based on three kinds of topologies, *serial*, *parallel*, and *dense* as shown in Figure



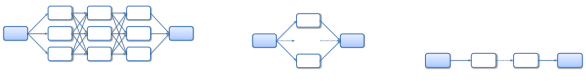
(a) Expression Size (b) Generation Time (c) Usage Time

Figure 5: Expression Size, Generation and Usage Time as Function of DB Size



(a) Expression Size (b) Generation Time (c) Usage Time

Figure 6: Expression Size, Generation and Usage Time as Function of FSM Size



(a) Dense (b) Parallel (c) Serial

Figure 7: Arctic Station Finite State Machines

7. The process specifications include queries with aggregates, and we have synthetically introduced external effect choices. The underlying real data consists of 25000 tuples and includes monthly meteorological observations, collected in 1961-2000. The number of nodes in the different process specifications (corresponding to stations) is at most 25. But to examine scalability with respect to the FSM size, we have also considered varying FSM sizes of up to 5000 nodes, following the topologies. For the *dense* structure we have varied both the fan-out and number of levels, and report results fan-out value of 10 and increasing number of levels.

6.2 Provenance Size

The first set of experiments aims at studying the size of obtained provenance expressions, as a function of the DDP size (state machine and underlying database sizes). In Figure 5a we present the expression size obtained using our two E-commerce datasets and varying the database size from 0 to $5M$ tuples. We observe a moderate growth of the provenance size with respect to growth of the database size, indicating the scalability of the approach (expression size of about 140MB and 180MB for database of 5M tuples, and for the two workflows). We note that the provenance size also depends (in an expected way) on factors such as the size of join results for guarding queries. For this dataset the join result size was proportional to the input DB size. We have also tried varying the join result size and observed the expected approximately linear dependency of provenance size with respect to it.

In Figure 6a, we have examined the effect of the *state machine size* on the provenance expression size. For that we have used the Arctic Stations dataset. The figure shows the results of the three topologies, for number of nodes that is increased up to 5000 (i.e. up to 200 times the real size). The expressions are compactly represented (this is made possible based on the congruence axioms) allowing for scalability even for the dense structure. Recall that the theoretical bound guarantees only exponential bound in this respect. Our experiments indicate that for relatively simple structures commonly found in workflows, the exponential bound is not met and feasibly small expressions are obtained.

6.3 Provenance Generation Time

The second set of experiments aims at assessing the *time* it takes to generate the provenance expression, again as a function of the DDP size. Figures 5b and 6b present the time it takes to generate the expression with respect to increased database and FSM sizes (respectively). Figure 5b presents the generation time of the provenance expression, on the two E-commerce DDPs, with database of increasing size from 0 to 5M tuples. Observe that the generation time is just under 35 and 25 seconds for DB size of 5M tuples (for the two DDPs). Recall that the provenance generation is done offline, rendering this computation time very reasonable. Figure 6b shows provenance generation time as a function of FSM size. For the "serial" and "parallel" structures, generation time is very fast even for very large FSMs. For the complex "dense" structure, the generation time is significantly slower due to the complex graph structure and

the needs for simplification, but is still under 1.5 minutes for a very large FSM with 5000 states.

6.4 Using Provenance Information

The last set of experiments aims at assessing the usefulness of the approach: it studies the time it takes to use the expression for observing results under hypothetical scenarios. In particular we have considered deletion propagation, and compared the observed times with those obtained through a simple baseline approach (the "competitor") of applying the deletions directly to the DDP and performing optimized temporal analysis of the obtained specification.

The results are reported in Figures 5c and 6c for growing DB size and FSM size respectively. Fig. 5c shows that using the provenance expression significantly outperforms the baseline approach. For DB of 5M tuples, the gain is about 65% for the benchmark DDP that follows [21], and 53% for our running example DDP. The results also indicate scalability of the approach, allowing to use the expression in about 2.5 and 3.5 seconds for the benchmark DDP and our running example (resp.), for DB size of 5M tuples (and much faster for smaller databases). We note that for our examples, all input tuples are reflected in the provenance. Our approach performance, in all measures *and specifically usage time and gain with respect to the competitor*, improves significantly as the percentage of such tuples drops (e.g. in a database with higher join selectivity).

The results in Fig. 6c indicate that for FSM structures commonly found in context of workflows, application of homomorphism to the computed expression is very fast. It was instantaneous for all structures of the Arctic Stations dataset (with their actual sizes). Even when extending the FSM size to up to 5000 nodes, it required less than 2.8 seconds for the dense structure, 1 second for the parallel structure and 0.8 seconds for the serial one. With respect to the competitor, a very significant gain was achieved for the dense and parallel structure (about 96% and 58% improvement for 5000 nodes). For the simple serial structure, there was no significant gain; this is due to the extreme simplicity of the FSM structure where the pre-processing effect diminishes.

7. RELATED WORK

Data Provenance. Provenance for data transformations (see e.g. [15, 7, 5, 23, 6, 8]) was shown useful in a variety of applications, including access control for data management, provisioning for database queries [12], and automatic optimization based on hypothetical reasoning for such queries [31, 30]. Unique technical challenges in our settings include accounting for the workflow in addition to data (including treatment of possibly infinitely many executions) in conjunction with the combination of different kinds of choices (external and data-dependent). The work of [28] has studied combination of annotations, but especially in the context of dependency between them and only for the relational algebra; in particular the issues that led us to consider tensor products are not addressed.

W3C Prov. W3C PROV-family [37] is a specification of standard for modeling and querying provenance for the web. Applications include explanations, assessments of quality, reliability or trustworthiness of data item, etc. The standard describes a range of transformations for data on the web, such as data generation, use and revision by multiple agents. We note that provenance for the analysis of possible executions of a data-dependent process, is studied here for the first time (through a precise algebraic notion). It is interesting to explore what aspects described in the W3C standard may be modeled through the semiring framework. The aspects that "fit" the framework, may possibly also be incorporated as part of the PADDP model, and perhaps accounted for in the analysis. For instance, trustworthiness may be captured by elements of the tropical semiring (see [22]); joint or alternative use of data by different agents may be represented by the semiring $\cdot$ or $+$ operations. Incorporating into the semiring framework other concepts of the W3C standard, for example specialization /inheritance and multiple agents, requires further work.

Workflow Provenance. Different approaches for capturing workflow provenance appear in the literature (e.g. [10, 9, 2, 26, 16, 35, 32]), but none of these models capture *semiring-based provenance* for *temporal analysis results* of workflow executions, and thus does not allow for the applications exemplified here such as provisioned temporal analysis. Semiring-based provenance for executions of data-centric workflows was studied in [3], however the development there (1) tracks provenance along with the execution rather than supporting analysis over all possible executions, (2) assumes a DAG-based control flow (no cycles), and (3) does not capture provenance for external effects.

Process Analysis. Temporal (and specifically LTL) analysis of (non-data-dependent) processes was studied extensively (see e.g. [29] for an overview), and several works have laid foundations for parameterization of such analysis (e.g. [27, 11]). However in contrast to our work, the process models in these works do not involve SQL-like queries to an underlying database. Analysis of *data-centric* processes was also studied in e.g. [9, 14, 18, 1], but *no provenance model was proposed there*, since the focus was on analyzing a concrete given process, rather than accounting for different weighting and for hypothetical changes as in our work. The process model in some of these works is richer than that of DDPs, mainly due to support of dynamic data updates and parallelism. Developing provenance support for these two features is an intriguing direction for future work.

8. CONCLUSION

We have presented in this paper a semiring based provenance framework for temporal analysis of data-dependent processes. We have studied properties of the framework and have demonstrated its usefulness.

As usual, there is a tradeoff between process model expressivity and complexity of process analysis (and provenance modeling). The DDP model is quite expressive, as it is based on an abstract control flow model (FSMs), with no restriction on the FSM states or topology. The expressive positive relational algebra with aggregates is used for capturing data dependency. We believe that our modeling choice constitutes a reasonable tradeoff between expressivity and analysis tractability. This is indicated by our implementation and experiments: we have shown that the model allows description of realistic and useful processes, as well as their efficient provenance support and provisioned analysis.

We note that the framework is "compositional": any query language for which one can compute semiring-based provenance can be "plugged-in" to the model instead of the one currently used. This may allow extensions, to capture an even larger class of data dependent process specifications.

Still, there are additional aspects of data-dependent processes, for which provenance modeling requires further work. We believe that the foundations laid in this paper will serve as sound grounds for the incorporation of provenance support for such features. In particular, an important feature in data-dependent process models is that of dynamic data updates. A sound support of (semiring-based) provenance for data updates, even when restricted to the context of relational algebra, is still the subject of ongoing research. As future work, we intend to study such modeling and its incorporation within our framework.

Another intriguing and important issue is that of parallel process executions. Our model for execution is a sequential one (an execution is essentially a path in the DDP). Still, one may capture some form of parallelism, for instance by designing FSM states that jointly capture the states of parallel threads. Once such an FSM simulating the behavior of a parallel process is designed, we may also capture properties related to e.g. scheduling in LTL (See e.g. [29]). Then, to simulate different scenarios, one may parameterize transitions and/or data items in a manner similar to that we have exemplified, to obtain a PADDP. Then, our evaluation algorithm may be executed to compute a provenance expression which in turn will allow the exploration of these scenarios, again in a manner similar to that exemplified. This approach captures essentially what is known in concurrency as interleaving semantics and it would be interesting to investigate other models of concurrency, e.g. Petri Nets, or ones involving fairness constraints. Building on the model and results in this paper, we intend to pursue this challenge in future work. Additional challenges for future work include the support of richer temporal logic formalisms (e.g, μ-calculus), and further optimizations.

9. REFERENCES

[1] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *PODS*, 1998.

[2] A. Ailamaki, Y. E. Ioannidis, and M. Livny. Scientific workflow management by database management. In *SSDBM*, 1998.

[3] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen. Putting lipstick on pig: Enabling database-style workflow provenance. *PVLDB*, 5(4):346–357, 2011.

[4] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

[5] O. Benjelloun, A.D. Sarma, A.Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17, 2008.

[6] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4), 2008.

[7] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[8] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[9] D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3), 2009.

[10] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD*, 2008.

[11] C. Daws. Symbolic and parametric model checking of discrete-time markov chains. In *Theoretical Aspects of Computing-ICTAC 2004*, pages 280–294. Springer, 2005.

[12] D. Deutch, Z. G. Ives, T. Milo, and V. Tannen. Caravan: Provisioning for what-if analysis. In *CIDR*, 2013.

[13] D. Deutch, Y. Moskovitch, and V. Tannen. Propolis: Provisioned analysis of data-centric processes (demo). In *VLDB*, 2013. to appear.

[14] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. A system for specification and verification of interactive, data-driven web applications. In *SIGMOD Conference*, 2006.

[15] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.

[16] I. Foster, J. Vockler, M. Wilde, and A. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. *SSDBM*, 2002.

[17] J. N. Foster, T. J. Green, and V. Tannen. Annotated xml: queries and provenance. In *PODS*, 2008.

[18] X. Fu, T. Bultan, and J. Su. Wsat: A tool for formal analysis of web services. In *CAV*, 2004.

[19] D. Giammarresi, J. L. Ponty, D. Wood, and D. Ziadi. A characterization of thompson digraphs. *Discrete Applied Mathematics*, 134(13):317 – 337, 2004.

[20] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *ASE*, pages 412–416, 2001.

[21] M. Gillmann, R. Mindermann, and G. Weikum. Benchmarking and configuration of workflow management systems. In *CoopIS*, 2000.

[22] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Provenance in orchestra. *IEEE Data Eng. Bull.*, 33(3), 2010.

[23] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[24] H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. In *ICALP*, 2008.

[25] H. Gruber and M. Holzer. Provably shorter regular expressions from deterministic finite automata. In *DLT*, 2008.

[26] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Res.*, 34, 2006.

[27] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. *Linear parametric model checking of timed automata.* Springer, 2001.

[28] E. V. Kostylev and P. Buneman. Combining dependent annotations for relational algebra. In *ICDT*, 2012.

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

[30] A. Meliou, W. Gatterbauer, and D. Suciu. Reverse data management. *PVLDB*, 4(12), 2011.

[31] A. Meliou and D. Suciu. Tiresias: the database oracle for how-to queries. In *SIGMOD*, 2012.

[32] P. Missier, N. Paton, and K. Belhajjame. Fine-grained and efficient lineage querying of collection-based workflow provenance. In *EDBT*, 2010.

[33] J. J. Morais, N. Moreira, and R. Reis. Acyclic automata with easy-to-find short regular expressions. In *CIAA*, 2005.

[34] N. Moreira and R. Reis. Series-parallel automata and short regular expressions. *Fundam. Inform.*, 91(3-4), 2009.

[35] Y. L. Simhan, B. Plale, and D. Gammon. Karma2: Provenance management for data-driven workflows. *Int. J. Web Service Res.*, 5(2), 2008.

[36] J. D. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, 1989.

[37] Prov-overview, w3c working group note, 2013. http://www.w3.org/TR/prov-overview/.