PROPOLIS: Provisioned Analysis of Data-Centric Processes [*]

Daniel Deutch
Ben Gurion University

Yuval Moskovitch
Ben Gurion University

Val Tannen
University of Pennsylvania

ABSTRACT

We consider in this demonstration the (static) analysis of data-centric process-based applications, namely applications that depend on an underlying database and whose control is guided by a finite state transition system. We observe that analysts of such applications often want to do more than analyze a specific instance of the application's process control and database. In particular they want to interactively test and explore the effect on analysis results of different hypothetical modifications applied to the application's transition system and to the underlying database. To that end, we propose a demonstration of `PROPOLIS`, a system for PROvisioned PrOcess anaLysIS, namely analysis of data-centric processes under hypothetical modification scenarios. Our solution is based on the notion of a provisioned expression (which in turn is based on the notion of data provenance), namely an expression that captures, in a compact way, the analysis result with respect to all possible combinations of scenarios, and allows for their exploration at interactive speed. We will demonstrate `PROPOLIS` in the context of an online shopping application, letting participants play the role of analysts.

1. INTRODUCTION

Many real-life applications rely, in intricate ways, on an underlying database in their operation. For example, E-commerce applications such as that of Ebay [9] rely on a database for management of products, orders etc., which affects the possible execution of the application and its interaction with potential users. 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.

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 logic can be captured via a Finite State Machine whose states reflect web-pages and logical application states, and (part of) its transitioning is governed by queries on an underlying relational database. Now, an analyst may be interested in answering, among others, the following questions:

- (A1) What is the probability that a user would exit without viewing the daily discount deals?
- (A2) What is the minimal number of clicks allowing a user to view daily deals?
- (A3) What is the probability that a user would view product sub-categories for some category?

Such analysis tasks can be expressed in LTL (Linear Temporal Logic) [12] which is commonly used in the context of static analysis. To obtain a better understanding of the application, analysts may want to do more than pose a single, static analysis task with respect to its current specification, but rather to test and explore the effect of different modifications to the process or underlying data, on analysis results. Hence, she may want to perform analysis of the above flavor under hypothetical scenarios such as:

- (S1) What if we remove/add a link between two web pages?
- (S2) What if a particular product is no longer available?
- (S3) What if the user is a member of a particular club?

In realistic cases, there may be many possible scenarios and *combinations thereof* that are of interest. An analyst would like to interactively explore the different combinations, refining them according to the analysis results (e.g. removing one link at a time and observing is effect). A simple way to support the exploration of such scenarios is to apply each requested hypothetical scenario to the application specification, and then analyze the obtained specification. This, however, would be *prohibitively inefficient*, as it involves repeated costly evaluation of the LTL formula as well as costly accesses to the application's underlying database.

Instead, we propose an alternative approach and implement it in `PROPOLIS` (*PROvisioned PrOcess anaLysIS*), whose high-level architecture is depicted in Fig. 1. Once the model specification was set, the analyst may (1) pose an LTL query and (2) define hypothetical scenarios by defining *parameters* with respect to the process specification (logical flow and/or underlying database) at points of interest. Then, `PROPOLIS` evaluates the given LTL formula with respect to the parameterized specification, and computes an expression referred to as a *provisioned expression.* This expression captures, in

[*]This research was partially supported by the Frankel Institute, the National Science Foundation (NSF IIS 1217798) and the US-Israel Binational Science Foundation

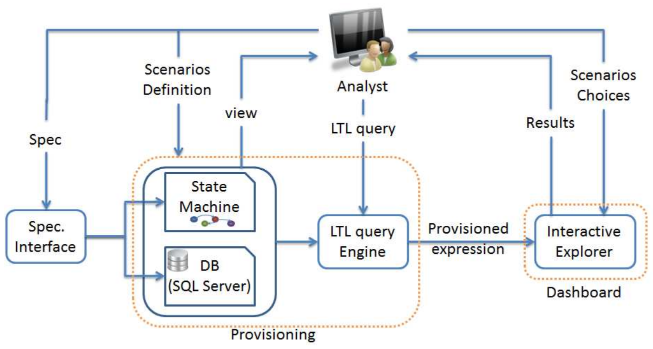

Figure 1: System Architecture

a compact way, the result of evaluating the LTL formula with respect to *applying all possible combinations of specified scenarios (parameter values) to the process specification.* The provisioned expression is passed to a module which allows interactive exploration of scenarios using only the provisioned expression while avoiding further costly access to the database or costly evaluation of the LTL formula.

We will demonstrate that `PROPOLIS` has several desired features. In particular:

- `PROPOLIS` allows to specify hypothetical scenarios of interest via a simple interface.
- `PROPOLIS` computes provisioned expression with reasonable overhead in time, w.r.t. LTL evaluation with respect to the original specification.
- The size of the provisioned expression is feasibly small. In particular, it depends on the size of the Finite State Machine and the number of scenario parameters, but is independent of the size of the underlying database.
- `PROPOLIS` allows for interactive and intuitive exploration of analysis results under hypothetical scenarios, which is faster by orders of magnitude than can be achieved by re-evaluation of the LTL formula on modified specifications.

The main technical challenges towards achieving those desiderata are related to computation of feasibly small provisioned expression that compactly captures properties of possibly infinitely many [1] executions, under exponentially many combinations of scenarios. To this end we devise a novel algorithm that leverages, combines and extends (1) algorithms for the analysis of (non-data-dependent) processes, allowing to compactly handle infinitely many executions, and (2) advancements in the management of *data provenance* based on the semiring framework [11, 2]. A provisioning paradigm in the context of analytical database queries was introduced in [7] but no provisioned data-centric process analysis was considered until now.

We next provide details on the technical background underlying `PROPOLIS` (Sec. 2) and on implementation issues (Sec. 3), and describe the demonstration scenario (Sec. 4).

2. TECHNICAL BACKGROUND

We (informally) introduce the main technical notions involved in the development of `PROPOLIS`, through examples.

[1]When there are loops in the Finite State Machine

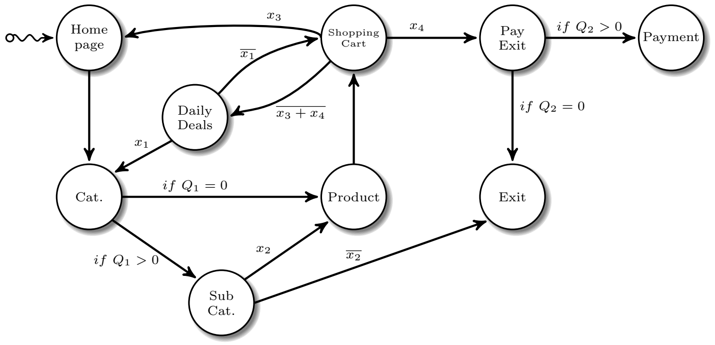

Figure 2: Data-Driven Process

Model. A data-dependent processes (DDPs) is defined by a Finite State Machine, where some transitions may be "guarded" by a boolean query on an underlying database. We support, for guarding queries, an expressive fragment of SQL, equivalent to *boolean positive relational algebra with aggregates.* Boolean queries results decide guarded transitions, while other transitions are decided by external factors such as user choices. A DDP execution is then a (finite or infinite) path in the state machine, and the set of all executions of a DDP S is denoted $exec(S)$.

EXAMPLE 2.1. *Consider the (partial) process logic in Figure 2. Each state intuitively stands for a web page, and transitions model links. Some transitioning is based on the user decision, such as the one from "shopping Cart" to "Pay and Exit", "HomePage" or "Daily Deals" depending on the user navigation choice (ignore for now the variables next to these transitions). Other transitions 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 subcategories) or to a "Product" page (listing relevant products) depend on availability of sub-categories. Such transitions are associated with boolean queries on an underlying database, such as "if $Q_1 = 0$" (checking for the existence of a subcategory of the chosen category). A fragment of the database and some simplified guarding queries are given in Figs. 3 (ignore for now the Prov. column) and 4 resp.*

Linear Temporal Logic (LTL). LTL allows formulating properties of (possibly infinitely long) executions. An LTL formula comprises of propositional formulas (possibly using AND,OR,NOT) w.r.t. a given process state (e.g. the state name), the logical modalities $X\ \phi$ standing for "ϕ must hold in the next state" and $\phi\ U\ \psi$ ("ϕ must hold until we reach a state where ψ holds"). Derived modalities include $F\ \phi$ (finally ϕ is true) and $\phi\ B\ \psi$(ϕ holds before ψ holds). E.g. the property "A user exiting without viewing the daily deals" (w.r.t. our example) is captured by the LTL formula *(Exit OR PayExit) B DailyDeals.* We use $e \models f$ to denote that an execution e satisfies an LTL formula f.

Provenance semirings. We rely for provisioning on semiring-based provenance, which we next briefly overview. 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 *commutative semiring* is then

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

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

Figure 3: Underlying Database

Q_1 :

```
SELECT COUNT(*)
FROM CategoryHierarcy CH,
SelectedCat SC
WHERE CH.Cat = SC.Cat
```

Q_2 :

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

Figure 4: SQL Queries

$(K, +_K, \cdot_K, 0_K, 1_K)$ 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 deep connection between the operations and axioms of the relational algebra and that of commutative semirings was established in [11]: intuitively, $\cdot_K$ can stand for a joint use of tuples (as in the join operation), while $+_K$ can stand for alternative use (as in projection). Since we will have to deal with infinite objects (infinitely many executions), we will consider here only ω-continuous semirings (see [11]). In particular such semirings satisfy that for any $a \in K$ it holds that $a^* = \sum_{i=0\ldots\infty} a^i$ (Kleenee star of a) is also an element of K. To support arbitrary aggregates for guarding queries we use a further construction whose details can be found in [2] but are omitted here for lack of space.

Provisioning. To support provisioning, we allow the analyst to choose some transitions and tuples whose modification she would like to explore; consequently tuples are associated with annotations taken from some semiring K, and transitions (corresponding to external effects) with elements of a possibly different semiring K'. This allows to parameterize both. We call the result an *annotated DDP*.

For example, reconsider Fig. 2, and note now the annotations for external provenance (x_1-x_4). We use $\overline{x_i}$ to denote the complement of x_i; i.e. if x_i is mapped to a boolean value, then $\overline{x_i}$ is mapped to its negation. Similarly, database tuples are annotated in Fig. 3 with d_1-d_4.

Next, we define two notions of provenance with respect to such executions. The *data provenance* of an execution e (with respect to an annotated DDP) is obtained as the accumulation (via product, following the idea of joint use), of provenance expressions obtained from evaluating the queries guarding transitions that were followed in the execution. This expression can be computed based on [11, 2]. Similarly, the *external provenance* is the product of provenance encountered in unguarded transitions (capturing external effects). The provenance of an execution, denoted $prov(e)$, is then defined as the pair (k, k') where k is the data provenance and k' is the external provenance of the execution. The intuitive interpretation is joint use of k and k' and thus we overload notation and use $k \cdot k'$ to denote the pair (k, k').

We are now ready to define provisioned expressions, via provenance. To that end, recall that summation corresponds to alternative computation; so, the provisioned expression for an LTL formula f with respect to an annotated DDP s (denoted $prov(f, S)$) is defined as the *formal sum* [11] of provenance of qualifying executions, i.e. $prov(f, s) = \sum_{\{e \in exec(S) | e \models f\}} prov(e)$. Note that $prov(f, S)$ is defined as a formal, possibly infinite, sum. However, we can compute a compact representation for it, intuitively resembling a regular expression of all executions conforming to the formula, capturing both data and external provenance of all such executions. The size of this expression is dependent only on the number of states and number of parameters and, as we will demonstrate, is feasibly small.

EXAMPLE 2.2. *Consider scenario A1 from the Introduction. The LTL formula* (Exit OR PayExit) B DailyDeals. *The provisioned expression is a formal sum over all (infinitely many) possible executions satisfying it, which can be compactly represented as follows. The provenance for database queries is captured by expressions on the d_i's (e.g. for $Q_1 > 0$ we get $d_1 \cdot d_4$; for the join result to be non-empty both tuples need to be present). Now let $P = d_1 \cdot d_4 \cdot (\overline{x_2} + x_2 \cdot x_4) + \overline{d_1 \cdot d_4} \cdot x_4$ (P captures paths leading from HomePage to one of the exit states without looping), and let $P_1 = (\overline{d_1 \cdot d_4} + d_1 \cdot d_4 \cdot x_2) \cdot x_3$ (intuituively capturing one traversal of the FSM loop). The provenance expression obtained is $P \cdot P_1^*$. We have obtained an expression that depends only on x's and d's. A1 asks for the* probability *of an event so we can "instantiate" the variables x_1-x_4 with probability values (assumed for simplicity to be independent), say* $0.7, 0.6, 0.5, 0.4$ *resp. To analyze the probability of the event with respect to the current process structure and underlying database, we simply set d_1-d_4 to* 1 *(and $\overline{d_i}$ to* 0*), i.e. corresponding tuples are all in the database. In this case we get (by simplification)* $0.64 \cdot 0.3^* \approx 0.91$ *(for the last step we have applied $a^* = \frac{1}{1-a}$ which holds for probabilities). What-if scenarios can be explored by e.g. assigning variables to* 0 *or* 1. *There are 2^8 such assignments, and analysts can, at interactive speed, explore those that are of interest to them. For instance, to re-compute the probability for the case that sub-categorization of "cell phones" is canceled (i.e. the corresponding tuple is omitted) we simply set $d_4 = 0$ to get that the probability has decreased to* $0.4 \cdot 0.5^* = 0.8$. *This computation involves only arithmetics.*

The same abstract expression can be used for "minimal number of steps". For that, we use the tropical *semiring [11] (+ is the standard min, $\cdot$ is the standard +). Hypothetical modifications can be examined simply by changing variables values.*

Related Work. Data-centric processes and workflows have been extensively studied in the database community, with research on e.g. management and analysis of *records* of executions (e.g. [1, 5]), simulation (e.g. [14]), or on static analysis of possible executions (e.g. [3, 8, 10, 4]). The latter is the most relevant to our work, but we are not aware of previous work allowing for *provisioned* static analysis of the flavor implemented in PROPOLIS. In the verification community there are works on parameterized process analysis (see e.g. [6]), but these do not consider processes that interact with an *underlying database*, and are thus not applicable to our settings. We have mentioned the previous work on provisioning in [7]. The complementary problem of automatic optimization based on hypothetical reasoning was studied in [13] for database queries, but not for process analysis.

3. IMPLEMENTATION ISSUES

PROPOLIS is implemented in C♯ with WPF GUI using .NET framework, and runs on Windows 7. It uses MS SQL server as its underlying database management system. We have presented the system architecture in the Introduction, and we next review some more implementation issues.

Provisioning. The provisioning module includes a generic implementation of a DDP model and of the analysis engine. Its input is an annotated DDP and an LTL formula. The DDP (without annotation) structure is fed once by an administrator, and the analyst specifies hypothetical scenarios with respect to it, resulting in an annotated DDP. The *specification of scenarios* is done via an intuitive interface: the analyst is shown the finite state machine structure as well as the underlying database, and by clicking can choose a transition (or a pair of states between which there is currently no transition), a database tuple or cell, or a group thereof. This leads to a pop-up where the analyst can choose to parameterize the selection (and specify an optional parameter name) or set a particular value where applicable. For the demonstration, we have designed a simple graphical interface for some common formulas (including those exemplified in this proposal), in addition to supporting arbitrary LTL formulas. The output of evaluating the LTL formula f on the annotated DDP S is a provisioned expression $prov(f, S)$ which is fed to the dashboard module.

Dashboard. Once a provisioned expression has been computed and fed to the dashboard module, the analyst can interact with this module through an interface allowing to change the value of the predefined parameters and observe the effect of the change on the result of the LTL formula. The dashboard interface is simple and intuitive: the parameter names and current values (with default values reflecting the current application structure and database state) are presented in a table. By clicking on a parameter name, the analyst can view the relevant part of the DDP specification / database which becomes highlighted. The analyst can then repeatedly change the value of any parameter, and view the analysis results with respect to the modification, which are computed at interactive speed.

4. DEMONSTRATION SCENARIO

We will demonstrate the usefulness of PROPOLIS in the context of an online shopping application, mimicking a portion of the Ebay web-site [9]. The logical flow and database schema for this demonstration were manually constructed based on the web-site and the database was populated based on observed data, combined with some (randomly generated) synthetic data (in realistic cases the analyst is likely to be given access to such data). We will highlight different parts of the model. We will then consider provisioned analysis. For the demonstration, we will pre-define the analysis tasks exemplified in this proposal and will allow a volunteer participant to choose between them (and to “tweak” them), or to pose other LTL formulas. We will also ask the participant to identify points in the process logic and/or underlying data whose modification they wish to explore, and to parameterize these through our interface. PROPOLIS will then compute a provisioned expression, demonstrating feasibility of this computation in terms of execution time.

Then, we will allow the participant to interactively explore modifications using the dashboard, based on the parameters she has defined. For each modification, the analysis results will be promptly computed and presented. This will demonstrate the intuitive and interactive nature of the exploration. During the exploration of scenarios by the participant, the system will record (for demonstration purposes) her choices. Then, we will show the audience a simulation of an alternative computation that repeats the LTL computation for every modified version of the process specification (obtained by “replaying” the participant choices). We will compare the average execution time of such evaluation to (1) the time it took PROPOLIS to compute the provisioned expression and (2) the average time it took PROPOLIS to compute the result following a modification, in the interactive process that was based on the expression. This will demonstrate that (1) computation of provisioned expression is comparable in terms of execution time to LTL evaluation and (2) exploration using PROPOLIS significantly outperforms re-evaluation of the formula. Last, we will allow the audience to look under the hood, showing and explaining the computed provisioned expression.

5. REFERENCES

[1] A. Ailamaki, Y. E. Ioannidis, and M. Livny. Scientific workflow management by database management. In *SSDBM*, pages 190–199, 1998.

[2] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.

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

[4] E. Damaggio, A. Deutsch, R. Hull, and V. Vianu. Automatic verification of data-centric business processes. In *BPM*, pages 252–267, 2011.

[5] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD*, pages 1345–1350, 2008.

[6] C. Daws. Symbolic and parametric model checking of discrete-time markov chains. In *ICTAC*, pages 280–294, 2004.

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

[8] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. A system for specification and verification of interactive, data-driven web applications. In *SIGMOD*, pages 772–774, 2006.

[9] http://www.ebay.com/.

[10] X. Fu, T. Bultan, and J. Su. Wsat: A tool for formal analysis of web services. In *CAV*, pages 510–514, 2004.

[11] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[12] Z. Manna and A. Pnueli. *Temporal Logic.* Springer, 1992.

[13] A. Meliou, Y. Song, and D. Suciu. Tiresias: a demonstration of how-to queries. In *SIGMOD*, pages 709–712, 2012.

[14] W. van der Aalst, J. Nakatumba, A. Rozinat, and N. Russell. Business process simulation: How to get it right? *BPM Center Technichal Report*, 2008.