Towards a Theory of Query Stability in Business Processes

Elisa Marengo, Werner Nutt, Ognjen Savković

Faculty of Computer Science
Free University of Bozen-Bolzano, Italy
firstname.lastname@unibz.it

1 Introduction

Data quality has attracted attention in theoretical database research in the past few years and different aspects, such as consistency, accuracy, currency, and completeness have been investigated [1–3]. One of the main factors that determine data quality is where and how data originate. We believe that analyzing how business processes generate data allows one to gather additional information on their fitness for use. Specifically, we want to understand whether an ongoing business process that reads from and writes into a database can affect the answer to a query or whether the answer is *stable*, that is, it will not change as a result of the process.

As motivating example consider the student registration at the University of Bozen-Bolzano. In November the student office distributed a report showing the numbers of students enrolled in the offered courses. When comparing the numbers with those of the previous years, the Master in Computer Science (MScCS) showed a decrease, in contrast with other courses, like the Master in Economics (MScECO), that registered a substantial increase. The reason for this discrepancy was a complication in the registration process, which foresees two routes to registration: an ordinary one and a second one via international federated study programs to which some Bolzano courses, like the MScCS, are affiliated. Due to different deadlines, ordinary registration was concluded in November while registration for students from federated programs was not. Since the MScCS is affiliated to some federated programs, but the MScECO not, the query asking for all MScECO students was stable in November and returned a reliable figure, while the query for all MScCS students was not and returned too low a number.

Even though in general a database may be constantly updating, and thus making data unstable, certain queries may have stable answers, (at least for some period of time). Registration at our University follows strict rules and is supported by an information system. If not only the data of the registration process were explicitly available, but also the rules, such a stability analysis of query answers could potentially be automated.

Assuming that data are created and manipulated according to a given business process, a formal reasoning task is to determine whether in all possible executions of such a process, the answer to a given query will remain the same. Then we say that the *query is stable*, and therefore reliable.

In this work, we propose a simple yet expressive formalism to model business processes that read, create and write data in an underlying database. We leverage the formal definition of such a business process to deduce whether a query answer is stable in case

the data is managed according to the rules of the process. We establish exact complexity measures for checking the stability of conjunctive queries in several variants of processes. Since our upper complexity bounds stem from reductions to the evaluation of certain FOL and Datalog queries, our work provides an immediate way for implementations using technologies such as SQL and ASP engines.

Related Work. Traditional approaches to business processes modeling are activity-centric and are based on (high-level) Petri Nets [4] and standards such as BPMN and BPEL. These approaches do not capture the specification of a database or the interaction with it. In recent years, the modeling of data and business processes under the same umbrella has gained significant attention [5–8]. Our work can be considered as a restricted case of [7, 8] where only database insertions are allowed but not updates or deletions. Checking properties of processes that allow unboundedly many data is inherently undecidable, and decidability is obtained by imposing additional restrictions (e.g., see *state boundedness* in [8]). As a difference, having insertions only allowed us to establish new decidable cases without additional restrictions. In the context of databases, query stability can be related to the problem of queries independent from updates [9, 10], i.e. checking when a query is independent from a set of updates over the database, by considering the update rules but not the database instance.

2 Data-aware Business Processes

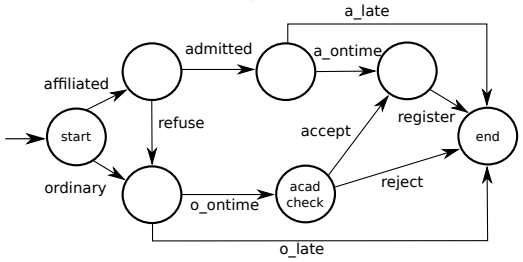
With our model of *data-aware business processes* (DABP) we want to capture some elementary aspects of how data are manipulated by business processes. A DABP over a schema Σ is a pair $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$, consisting of a *process part* $\mathcal{P}$ and a *configuration part* $\mathcal{C}$. Intuitively, the process part is fixed. It defines how and under which conditions actions can change data stored in the configuration part. The configuration part comprises an instance of the underlying database and the state of all active process instances.

Process Part. The skeleton of the process part is a directed potentially cyclic graph $N = \langle P, T \rangle$, the *process net*, consisting of a set of vertices P, the *places*, and a set of edges T, the *transitions*. There is one distinguished place in P, the start place *start*. There is also a distinguished relation symbol, I, the input of a process instance. The last argument of I is a timestamp τ, called the *start time*, to record the time when the process instance was started. We denote the schema Σ augmented by I as Σ_I. In a process instance, an object holding an I-atom traverses the graph, starting from *start*. Thus, the different transitions emanating from a place represent alternative developments of a process instance.

The whole process part is a pair $\mathcal{P} = \langle N, L \rangle$, which in addition to a network N comprises a *labeling function* L that assigns to every transition $t \in T$ a pair $L(t) = (E_t, W_t)$. Here, E_t, the *execution condition,* is a Boolean query over Σ_I and W_t, the *writing rule,* is a rule $Q_t(\bar{x}) \to R(\bar{x})$ whose head is a relation of Σ and whose body is a Σ_I-query that has the same arity as the head relation. Evaluating W_t over a Σ-instance D results in the set of ground atoms $W_t(D) = \{R(\bar{c}) \mid \bar{c} \in Q_t(D)\}$. Intuitively, E_t specifies in which state of the database which object can perform the transition t and W_t specifies which new information is (or can be) written into the database when

Table 1. *(a)* Graphical representation of the DABP process net for student registration scenario, execution conditions and writing rules. *(b)* A Database instance for the reference scenario.

(a)



$E_{affiliated} = I(S,P,T), studyplan(P, affil, M)$
$E_{ordinary} = I(S,P,T), studyplan(P, ord, M), \neg studyplan(P, affil, M)$
$E_{admitted} = I(S,P,T), admitted(S,P)$
$E_{refuse} = I(S,P,T), \neg admitted(S,P), studyplan(P, ord, M)$
$E_{a\_late} = I(S,P,T), deadline(affil, D),\ T > D$
$E_{a\_ontime} = I(S,P,T), deadline(affil, D),\ T < D$
$W_{register} = I(S,P,T), studyplan(P,R,M) \rightarrow registered(S,M,P)$
$E_{o\_late} = I(S,P,T),\ deadline(ord, D),\ T > D$
$E_{o\_ontime} = I(S,P,T),\ deadline(ord, D),\ T < D$
$E_{register} = E_{accept} = E_{reject} = true$

(b)

studyplan

program	*registr.*	*master*
emSE	*affil*	mscCS
emCL	*affil*	mscCS
emCL	*ord*	mscCS
db	*ord*	mscCS
econ	*ord*	mscECO

admitted

student	*program*
bob	emCL
mary	emSE

deadline

registr.	*date*
ord	1^{st} Oct
affil	1^{st} Dec

registered

student	*master*	*program*
bob	mscCS	emCL

performing t. In this paper we assume that E_t and Q_t are conjunctive queries with negated atoms and possibly comparisons involving timestamps.

Example 1. Consider again the scenario described in the Introduction. Table 1*(a)* contains a graphical representation of a DABP process net for a simplified student registration process, together with execution conditions and writing rules.

A student who wants to register to a program needs to fill a form providing her *name* and the *program* she wants to apply to. When received by the administration, the request is associated with a *timestamp*. We represent this information by a ground atom $I(s, p, \tau)$ of the relation I. The available programs are of two kinds: those that are *affiliated* to international federated programs, and *ordinary* ones. For federated programs, an international commission decides about whom to admit and these decisions are stored in the table *admitted*. For ordinary programs, the university takes the decision.

According to this distinction, the first check in the process is to determine to which kind of program the request I refers to: affiliated ($E_{affiliated}$) or ordinary ($E_{ordinary}$). In the first case, a student who is already admitted to the federated program can proceed towards registration. Non-admitted students can go for an ordinary registration provided the program is open also to this kind of students (E_{refuse}). In case the student is admitted to the federated course, then the corresponding deadline for the application is looked up in the database: if the request arrived after the deadline (E_{a_late}) then the registration process ends. Otherwise, if the request arrived on time (E_{a_ontime}) the student is registered ($E_{register}$) and a corresponding atom is inserted into the database instance ($W_{register}$). Similarly, for a late ordinary request (E_{o_late}) the process is ended, while for a request arrived on time (E_{o_ontime}) the academic merits are checked (*acad check*). This human intervention is modelled as a non-deterministic choice that can result in the request being accepted (E_{accept}) or rejected (E_{reject}). If accepted, the student is registered.

Configuration Part. This part models the data that is manipulated by the process part. Formally, a configuration is a quadruple $\langle D, O, M, \tau \rangle$, where D is an instance of the schema Σ, O is a set of process instances, which we call *process objects*, M is a mapping that associates every object $o \in O$ with a place $M_P(o) \in P$ and with a ground I-atom $I(\bar{c}) = M_I(o)$, and τ is a timestamp, the current time. We assume that for all objects $o \in O$ the start time in $M_I(o)$ is less or equal than the current time.

Example 2. Table 1*(b)* shows a simplified database instance for our reference scenario. Relation *studyplan* stores the study programs offered by the university, the kind of registration they allow (ordinary or affiliated), and the master they belong to; *admitted* contains the students already admitted to a federated program; *deadline* stores the registration deadlines for the registrations to ordinary and affiliated courses; *registered* contains the students that successfully completed the registration process.

We consider an *input relation* I of arity 3, carrying the following information about the request: *(i)* the student name; *(ii)* the requested program; and *(iii)* the time of the request. In our example, there are no objects currently in the net.

Execution of DABP. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ be a DABP, with current configuration $\mathcal{C} = \langle D, O, M, \tau \rangle$. There are two kinds of *atomic execution* steps of a DABP, *(i)* the *traversal* of a transition in the net by an object or *(ii)* the *introduction* of a new object.

Traversal of an enabled transition by an object. Consider an object $o \in O$ with $M(o) = (p_1, I(\bar{c}))$. That is, o is at place p_1 and $I(\bar{c})$ are the input data of o. Let t be a transition from p_1 to p_2, with execution condition E_t. Then we say that t is *enabled* for o if $D \cup \{I(\bar{c})\} \models E_t$. Let $W_t = (Q_t(\bar{x}) \rightarrow R(\bar{x})$ be the writing action of t. Then the effect of o traversing t is the transition from $\mathcal{C} = \langle D, O, M, \tau \rangle$ to a new configuration $\mathcal{C}' = \langle D', O, M', \tau \rangle$, such that *(i)* $D' = D \cup W_t(D \cup \{I(\bar{c})\})$ is the new database; *(ii)* the set of objects and the current time is the same, and *(iii)* M' is an update of M that reflects the change of place, that is, $M'(o) = (p_2, I(\bar{c}))$ and $M'(o') = M(o')$ for all other objects o'.

Introduction of an arbitrary object at the *start* place. Let o' be a fresh object and let $I(\bar{c}', \tau')$ be an atom where $\bar{c}'$ is a vector of constants, and the timestamp τ' is greater or equal than $\tau_{\mathcal{C}}$, the current time of $\mathcal{C}$. Note that the constants in $\bar{c}'$ need not appear in the database or in the process. The result of introducing o' with info $\bar{c}'$ at time τ' is the configuration $\mathcal{C}' = \langle D, O', M', \tau' \rangle$, where *(i)* the database instance is the same as in $\mathcal{C}$; *(ii)* the set of objects $O' = O \cup \{o'\}$ has been augmented by o'; and *(iii)* the mapping M' is an extension of M to O', obtained by defining $M'(o') = (\textit{start}, I(\bar{c}', \tau'))$ and $M'(o) = M(o)$ for all $o \in O$.

An arbitrary *execution* is a sequence of atomic execution steps. Since for every configuration $\mathcal{C}$ one can introduce new objects at the start place, there are always several atomic executions possible for $\mathcal{C}$. We say that a configuration $\mathcal{C}'$ is *reachable* from $\mathcal{C}$ if there exists a finite sequence of atomic executions, such that $\mathcal{C}$ is the first and $\mathcal{C}'$ is the last configuration in the sequence.

Finally, we define the property of query stability.

Definition 1 (Query Stability). *Given a DABP $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ and a CQ Q. Then Q is* stable *in $\mathcal{B}$ if for any reachable configuration $\mathcal{C}' = \langle D', O', M', t' \rangle$ holds:*

$$Q(D) = Q(D').$$

We illustrate this property with our running example. Consider the queries $Q_{\text{cs}}(S) \leftarrow$ *registered*$(S, \textit{mscCS}, P)$ and $Q_{\text{eco}}(S) \leftarrow$ *registered*$(S, \textit{mscECO}, P)$ that ask for the students registered at the master in CS, and the master in Economics, respectively. Depending on the current time, one can analyze the stability of the two queries. If the current time is before the 1st of October both queries are unstable since arbitrary new students can register. If the current time is after the 1st of December, both queries are stable since the two deadlines have passed. When the current time is between two deadlines, Q_{eco} is stable because the deadline for ordinary programs has passed and *mscECO* is not affiliated to any program. On the other hand, Q_{cs} is not stable because it is affiliated to the program *emSE*, for which *mary* did not register yet. Note that if she was registered, Q_{cs} would be stable since all admitted student would be registered.

3 Reasoning about Query Stability in DABP

We investigated how to check whether a conjunctive query Q is stable in a DABP $\mathcal{B}$. To understand the possible sources of complexity, we studied several types of processes that differ in the way they interact with the database and the way data can be entered into the process.

The first distinction is whether the model allows a process object to read the facts that itself or another object has written into the database. In the general case, denoted DABP, this is allowed. The restricted case, denoted DABP$_{\textit{rowo}}$ (read-only write-only), does not allow this. Formally, it splits the schema Σ into disjoint schemas, the reading schema Σ_r and the writing schema Σ_w, such that the execution conditions and the queries in the writing rules range over Σ_r while the heads range over Σ_w. Our running example is in DABP$_{\textit{rowo}}$.

We consider a process under *open semantics*, where new process objects may start in any moment. Alternatively, we also consider a process under *closed semantics*, that is we only admit transition traversals as possible execution steps (no new instances can be started). In this case, stability of a query depends only on the unfinished objects, while in open processes, it depends also on new objects that may start.

We also distinguish the case in which the initial configuration of the process does not contain any object, called *fresh* DABP, from the *arbitrary* case in which we do not make any assumption on the presence or absence of objects. Notice that under closed semantics the only interesting case is the arbitrary one (in a closed DABP with a fresh configuration a query answer is trivially stable since no objects can be inserted).

Table 2 summarises the complexity of checking stability for the different cases. Due to the limited space, we provide a brief intuition of the results while omitting the technical details. In general case, DABP allows unboundedly many new objects (data values) which combined with negation allows us to encode halting problem. Decidability is obtained either by disallowing negation (cases in brackets) or by disallowing new objects (closed semantics). For decidable cases, we established a correspondence between checking stability and brave entailment in Datalog with negation under stable model semantics. Similarly, for DABPs without negation we established a correspondence with entailment in positive Datalog. In the case of DABP$_{\textit{rowo}}$, the complexity drops significantly due to the "non-recursive" rules. In particular, stability can be decided by FOL

Table 2. Computational complexity of checking stability in DABP for conjunctive queries. The measures are lower and upper bounds (only AC^0 is in). In general, the lower bounds already hold for the processes without negation in the rules, except for the cases shown in brackets.

Semantics & Init. Conf.	DABP				$DABP_{rowo}$			
	Data	Process	Query	Combined	Data	Process	Query	Combined
Open & Arbitrary	UNDEC. (coNP)	UNDEC. (EXPTIME)	Π_2^P	UNDEC. (EXPTIME)	in AC^0	coNP	Π_2^P	Π_2^P
Open & Fresh	UNDEC. (PTIME)	UNDEC. (EXPTIME)	Π_2^P	UNDEC. (EXPTIME)	in AC^0	coNP	Π_2^P	Π_2^P
Closed & Arbitrary	coNP (coNP)	coNEXPTIME (EXPTIME)	Π_2^P	coNEXPTIME (EXPTIME)	in AC^0	coNP	Π_2^P	Π_2^P

query evaluation. Given a $DABP_{rowo}$ and a query we are able to encode the process part and the query into a FOL query that evaluates to *true* over the configuration iff the original query is not stable. The result of query complexity follows from the fact that deciding whether the two answers of a conjunctive query over two databases are the same is Π_2^P-complete in the query size.

4 Future Work

In this work we investigated the problem of determining the stability of a query answer when data is manipulated by a business process. As future work we plan to develop the DABP formalism further in the following ways: *(i)* consider more expressive queries, e.g., $CQ^{\neg}$ or FO; *(ii)* consider stability of aggregate queries and introduce aggregates in the process rules; *(iii)* quantify instability (in case a query is not stable, compute the minimal and maximal number of possible new answers, e.g., newly registered students); *(iv)* consider data quality aspects such as *data timeliness* and *data currency*.

Acknowledgements. This work was partially supported by the projects MAGIC and RARE, funded by the Province of Bozen-Bolzano.

References

1. Cong, G., Fan, W., Geerts, F., Jia, X., Ma, S.: Improving data quality: Consistency and accuracy. In: VLDB. (2007) 315–326
2. Fan, W., Geerts, F., Wijsen, J.: Determining the currency of data. ACM Trans. Database Syst. **37**(4) (2012) 25
3. Razniewski, S., Nutt, W.: Completeness of queries over incomplete databases. PVLDB **4**(11) (2011) 749–760
4. van der Aalst, W.M.P.: Verification of workflow nets. In: ICATPN. (1997) 407–426
5. Abiteboul, S., Vianu, V., Fordham, B.S., Yesha, Y.: Relational transducers for electronic commerce. In: PODS. (1998) 179–187
6. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: BPM. (2007) 288–304
7. Deutsch, A., Sui, L., Vianu, V.: Specification and verification of data-driven web services. In: PODS. (2004) 71–82
8. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: PODS. (2013) 163–174
9. Elkan, C.: Independence of logic database queries and updates. In: PODS. (1990) 154–160
10. Levy, A.Y., Sagiv, Y.: Queries independent of updates. In: VLDB. (1993) 171–181