Query Stability in Monotonic Data-Aware Business Processes

Ognjen Savković[1], Elisa Marengo[2], and Werner Nutt[3]

1 **Free University of Bozen-Bolzano, Bolzano, Italy**
`ognjen.savkovic@unibz.it`
2 **Free University of Bozen-Bolzano, Bolzano, Italy**
`elisa.marengo@unibz.it`
3 **Free University of Bozen-Bolzano, Bolzano, Italy**
`werner.nutt@unibz.it`

Abstract

Organizations continuously accumulate data, often according to some business processes. If one poses a query over such data for decision support, it is important to know whether the query is *stable,* that is, whether the answers will stay the same or may change in the future because business processes may add further data. We investigate query stability for conjunctive queries. To this end, we define a formalism that combines an explicit representation of the control flow of a process with a specification of how data is read and inserted into the database. We consider different restrictions of the process model and the state of the system, such as negation in conditions, cyclic executions, read access to written data, presence of pending process instances, and the possibility to start fresh process instances. We identify for which restriction combinations stability of conjunctive queries is decidable and provide encodings into variants of Datalog that are optimal with respect to the worst-case complexity of the problem.

1998 ACM Subject Classification H.2.4 [Systems]: Relational databases

Keywords and phrases Business Processes, Query Stability

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.16

1 Introduction

Data quality focuses on understanding how much data is fit for its intended use. This problem has been investigated in database theory, considering aspects such as consistency, currency, and completeness [8, 13, 23]. A question that these approaches consider only marginally is where data originates and how it evolves.

Although in general a database may evolve in arbitrary ways, often data are generated according to some business process, implemented in an information system that accesses the DB. We believe that analyzing how business processes generate data allows one to gather additional information on their fitness for use. In this work, we focus on a particular aspect of data quality, that is the problem whether a business process that reads from and writes into a database can affect the answer of a query or whether the answer will not change as a result of the process. We refer to this problem as query stability.

For example, consider a student registration process at a university. The university maintains a relation *Active*(*course*) with all active courses and a table *Registered*(*student*, *course*) that records which students have been registered for which course. Suppose we have a process model that does not allow processes to write into *Active* and which states that before a student is registered for a course, there must be a check that the course is active.

Consider the query Q_{agro} that asks for all students registered for the MSc in Agronomics (*mscAgro*). If *mscAgro* does not occur in *Active*, then no student can be registered and the query is stable. Consider next the query $Q_{courses}$ that asks for all courses for which some student is registered. If for each active course there is at least one student registered, then again the query is stable, otherwise, it is not stable because some student could register for a so far empty active course.

In general, query results can be affected by the activities of processes in several ways. Processes may store data from outside in the database, e.g., the application details submitted by students are stored in the database. Processes may not proceed because data does not satisfy a required condition, e.g., an applicant cannot register because his degree is not among the recognized degrees. Processes may copy data from one part of a database to another one, e.g., students who passed all exams are automatically registered for the next year. Processes may interact with each other in that one process writes data that is read by another one, e.g., the grades of entry exams stored by the student office are used by academic admission committees. Finally, some activities depend on deadlines so that data cannot change before or after a deadline.

Approach. Assessing query stability by leveraging on processes gives rise to several research questions.

1. What is a good model to represent processes, data and the interplay among the two?
2. How can one reason on query stability in such a model and how feasible is that?
3. What characteristics of the model may complicate reasoning?

(1) Monotonic Data-Aware Business Process Model. Business processes are often specified in standardized languages, such as BPMN [22], and organizations rely on engines that can run those processes (e.g., Bonita [7], Bizagi [16]). However, in these systems how the data is manipulated by the process is implicit in the code. Current theory approaches either focus on *process* modeling, representing the data in a limited way (like in Petri Nets [18]), or adopt a *data* perspective, leaving the representation of the process flow implicit [6, 4, 11]. We introduce a formalism called Monotonic Data-aware Business Processes (MDBPs). In MDBPs the process is represented as a graph. The interactions with an underlying database are expressed by annotating the graph with information on which data is read from the database and which is written into it. In MDBPs it is possible that several process instances execute the process. New information (fresh data) can be brought into the process by starting a fresh process instance (Section 2). MDBPs are monotonic in that data can only be inserted, but not deleted or updated.

(2) Datalog Encodings. Existing approaches aim at the verification of general (e.g. temporal) properties, for which reasoning is typically intractable [4, 10, 11]. In contrast, we study a specific property, namely stability of conjunctive queries (Section 3), over processes that only insert data. This allows us to map the problem to the one of query answering in Datalog. The encoding generates all maximal representative extensions of the database that can be produced in the process executions and checks if any new query answer is produced. We prove that our approach is optimal w.r.t. worst case complexity in the size of the data, query, process model and in the size of the entire input.

(3) MDBP Variants. When modeling processes and data, checking properties often becomes highly complex or undecidable. While other approaches in database theory aim at exploring

the frontiers of decidability by restricting the possibility to introduce fresh data, we adopt a more bottom-up approach and focus on a simpler problem that can be approached by established database techniques. To understand the sources of complexity of our reasoning problem, we identify *five restrictions* of MDBPs:

(i) negation is (is not) allowed in process conditions;
(ii) the process can (cannot) start with pending instances;
(iii) a process can (cannot) have cycles;
(iv) a process can (cannot) read from relations that it can write;
(v) new instances can (cannot) start at any moment.

We investigate the stability problem for each combination of the restrictions above, called variants (Sections 3–9).

Related work and conclusions end the paper (Sections 10, 11). A technical report, with complete encodings and proofs can be found in [24].

A preliminary version of this paper was presented at the AMW workshop [21].

2 Monotonic Data-Aware Business Processes

Monotonic Data-aware Business Processes (MDBPs) are the formalism by which we represent business processes and the way they manipulate data. We rely on this formalism to perform reasoning on query stability.

Notation. We adopt standard notation from databases. In particular, we assume an infinite set of relation symbols, an infinite set of constants dom as the *domain of values*, and the positive rationals $\mathbb{Q}^+$ as the *domain of timestamps*. A schema is a finite set of relation symbols. A *database instance* is a finite set of ground atoms, called *facts*, over a schema and the *domain* $dom_{\mathbb{Q}^+} = dom \cup \mathbb{Q}^+$.We use upper-case letters for variables, lower-case for constants, and overline for tuples, e.g., $\bar{c}$.

An MDBP is a pair $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$, consisting of a *process model* $\mathcal{P}$ and a *configuration* $\mathcal{C}$. The process model defines how and under which conditions actions change data stored in the configuration. The configuration is dynamic, consisting of

(i) a database,and
(ii) the process instances.

Process Model. The process model is a pair $\mathcal{P} = \langle N, L \rangle$, comprising a directed multi-graph N, the *process net*, and a *labeling function* L, defined on the edges of N.

The net $N = \langle P, T \rangle$ consists of a set of vertices P, the *places*, and a multiset of edges T, the *transitions*. A process instance traverses the net, starting from the distinguished place *start*. The transitions emanating from a place represent alternative developments of an instance.

A process instance has input data associated with it, which are represented by a fact $In(\bar{c}, \tau)$, where In is distinguished relation symbol, $\bar{c}$ is a tuple of constants from $dom_{\mathbb{Q}^+}$, and $\tau \in \mathbb{Q}^+$ is a time stamp that records when the process instance was started. We denote with $\Sigma_{\mathcal{B},In}$ and $\Sigma_{\mathcal{B}}$ the schemas of $\mathcal{B}$ with and without In, respectively.

The labeling function L 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 $\Sigma_{\mathcal{B},In}$ and W_t, the *writing rule,* is a rule $R(\bar{u}) \leftarrow B_t(\bar{u})$ whose head is a relation of $\Sigma_{\mathcal{B}}$ and whose body is a $\Sigma_{\mathcal{B},In}$-query that has the same arity as the head relation. Evaluating W_t over a $\Sigma_{\mathcal{B},In}$-instance $\mathcal{D}$ results in the set of facts $W_t(\mathcal{D}) = \{R(\bar{c}) \mid \bar{c} \in B_t(\mathcal{D})\}$. Intuitively, E_t specifies in which state of the database

which process instance can perform the transition t, and W_t specifies which new information is (or can be) written into the database when performing t. In this paper we assume that E_t and B_t are conjunctive queries, possibly with negated atoms and inequality atoms with "$<$" and "$\leq$" involving timestamps. We assume inequalities to consist of one constant and one variable, like $X < 1^{\text{st}}$ Sep. We introduce these restricted inequalities so that we can model deadlines, without introducing an additional source of complexity for reasoning.

Configuration. This component models the dynamics of an MDBP. Formally, a configuration is a triple $\langle \mathcal{I}, \mathcal{D}, \tau \rangle$, consisting of a part $\mathcal{I}$ that captures the process instances, a database instance $\mathcal{D}$ over $\Sigma_{\mathcal{B}}$, and a timestamp τ, the current time. The instance part, again, is a triple $\mathcal{I} = \langle O, M_{In}, M_P \rangle$, where $O = \{o_1, ..., o_k\}$ is a set of objects, called *process instances*, and M_{In}, M_P are mappings, associating each $o \in O$ with a fact $M_{In}(o) = In(\bar{c}, \tau)$, its input record, and a place $M_P(o) \in P$, its current, respectively.

The input record is created when the instance starts and cannot be changed later on. While the data of the input record may be different from the constants in the database, they can be copied into the database by writing rules. A process instance can see the entire database, but only its own input record.

For convenience, we also use the notation $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D}, \tau \rangle$, $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ (when τ is not relevant), or $\mathcal{B} = \langle \mathcal{P}, \mathcal{D} \rangle$ (for a process that is initially without running instances).

Execution of an MDBP. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ be an MDBP, with current configuration $\mathcal{C} = \langle \mathcal{I}, \mathcal{D}, \tau \rangle$. There are two kinds of *atomic execution* steps of an MDBP:

(i) the *traversal* of a transition by an instance and
(ii) the *start* of a new instance.

(i) Traversal of a transition. Consider an instance $o \in O$ with record $M_{In}(o) = In(\bar{c}, \tau')$, currently at place $M_P(o) = q$. Let t be a transition from q to p, with execution condition E_t. Then t is *enabled for o,* i.e., o can *traverse* t, if E_t evaluates to *true* over the database $\mathcal{D} \cup \{In(\bar{c}, \tau')\}$. Let $W_t \colon R(\bar{u}) \leftarrow B_t(\bar{u})$ be the writing rule of t. Then the effect of o traversing t is the transition from $\mathcal{C} = \langle \mathcal{I}, \mathcal{D}, \tau \rangle$ to a new configuration $\mathcal{C}' = \langle \mathcal{I}', \mathcal{D}', \tau \rangle$, such that

(i) the set of instances O and the current time τ are the same;
(ii) the new database instance is $\mathcal{D}' = \mathcal{D} \cup W_t(\mathcal{D} \cup \{In(\bar{c}, \tau')\})$, and
(iii) $\mathcal{I} = \langle O, M_{In}, M_P \rangle$ is updated to $\mathcal{I}' = \langle O, M_{In}, M'_P \rangle$ reflecting the change of place for the instance o, that is $M'_P(o) = p$ and $M'_P(o') = M_P(o')$ for all other instances o'.

(ii) Start of a new instance. Let o' be a fresh instance and let $In(\bar{c}', \tau')$ be an In-fact with $\tau' \geq \tau$, the current time of $\mathcal{C}$. The result of starting o' with info $\bar{c}'$ at time τ' is the configuration $\mathcal{C}' = \langle \mathcal{I}', \mathcal{D}, \tau' \rangle$ where $\mathcal{I}' = \langle O', M'_{In}, M'_P \rangle$ such that

(i) the database instance is the same as in $\mathcal{C}$,
(ii) the set of instances $O' = O \cup \{o'\}$ is augmented by o', and
(iii) the mappings M'_{In} and M'_P are extensions of M_{In} and M_P, resp., obtained by defining $M'_{In}(o') = In(\bar{c}', \tau')$ and $M'_P(o') = start$.

An *execution* Υ of $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ is a finite sequence of configurations $\mathcal{C}_1, \ldots, \mathcal{C}_n$

(i) starting with $\mathcal{C}$ $(= \mathcal{C}_1)$, where
(ii) each $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ by an atomic execution step.

We denote Υ also with $\mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_n$. We say that the execution Υ *produces* the facts $A_1, \ldots, A_n$ if the database of the last configuration $\mathcal{C}_n$ in Υ contains $A_1, \ldots, A_n$. Since at each step a new instance can start, or an instance can write new data,

Table 1 Computational complexity of query stability in MDBPs. The results in a row hold for the class of MDBPs satisfying the defining restrictions and for the subclasses satisfying one or more of the optional restrictions. The results for all decidable variants indicate matching lower and upper bounds (except for AC^0). The $*$ indicates that the results for rowo hold for all non-trivial combinations of restrictions. All results for data, process, query and combined complexity of the decidable variants hold already for singleton MDBPs. †Note that, in all fresh variants instance complexity can be trivially decided in constant time (omitted in the table).

Defining Restrictions	Optional Restrictions	Data	Instance	Process	Query	Combined	Sect.
—	*fresh*†,*acyclic*	Undec.	Undec.	Undec.	Undec.	Undec.	4
closed	—	co-NP	co-NP	co-NExpTime	Π_2^p	co-NExpTime	6
positive	*closed*	PTime	co-NP	ExpTime	Π_2^p	ExpTime	5, 8
positive	*fresh*†, *acyclic*	PTime	co-NP	ExpTime	Π_2^p	ExpTime	8
closed, acyclic	*positive*	in AC^0	co-NP	PSpace	Π_2^p	PSpace	7
rowo	$*^\dagger$	in AC^0	in AC^0	co-NP	Π_2^p	Π_2^p	9

(i) there are infinitely many possible executions, and
(ii) the database may grow in an unbounded way over time.

3 The Query Stability Problem

In this section, we define the problem of query stability in MDBPs with its variants.

▶ **Definition 1** (Query Stability). Given $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$ with database instance $\mathcal{D}$, a query Q, and a timestamp τ, we say that Q *is stable in* $\mathcal{B}$ *until* τ, if for every execution $\mathcal{C} \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}'$ in $\mathcal{B}$, where $\mathcal{C}'$ has database $\mathcal{D}'$ and timestamp τ' such that $\tau' < \tau$, it holds that

$$Q(\mathcal{D}) = Q(\mathcal{D}').$$

If the query Q is stable until time point ∞, we say it is *globally stable*, or simply, *stable*.

The interesting question from an application view is: *Given an MDBP* $\mathcal{B}$*, a query* Q*, and a timestamp* τ*, is* Q *stable in* $\mathcal{B}$ *until* τ*?* Stability until a time-point τ can be reduced to global stability. One can modify a given MDBP by adding a new *start* place and connecting it to the old *start* place via a transition that is enabled only for instances with timestamp smaller than τ. Then a query Q is globally stable in the resulting MDBP iff in the original MDBP it is stable until τ.

To investigate sources of complexity and provide suitable encodings into Datalog, we identify five restrictions on MDBPs.

▶ **Definition 2** (Restriction on MDBPs and MDBP Executions). Let $\mathcal{B}$ be an MDBP.

Positive: $\mathcal{B}$ is *positive* if execution conditions and writing rules contain only positive atoms;
Fresh: $\mathcal{B}$ is *fresh* if its configuration does not contain any running instances;
Acyclic: $\mathcal{B}$ is *acyclic* if the process net is cycle-free;
Rowo: $\mathcal{B}$ is *rowo* (= read-only-write-only) if the schema Σ of $\mathcal{B}$ can be split into two disjoint schemas: the reading schema Σ_r and the writing schema Σ_w, such that execution conditions and queries in the writing rules range over Σ_r while the heads range over Σ_w;

Closed: an execution of $\mathcal{B}$ is *closed* if it contains only transition traversals and no new instances are started.

We will develop methods for stability checking in MDBPs for all combinations of those five restrictions. For convenience, we will say that an MDBP $\mathcal{B}$ is *closed* if we consider only closed executions of $\mathcal{B}$. A *singleton MDBP* is a closed MDBP with a single instance in the initial configuration.

Complexity Measures. The input for our decision problem are an MDBP $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$, consisting of a process model $\mathcal{P}$, an instance part $\mathcal{I}$, a database $\mathcal{D}$ and a timestamp τ, and a query Q. The question is: *Is Q globally stable in $\langle \mathcal{P}, \mathcal{I}, \mathcal{D}, \tau \rangle$?* We refer to process, instance, data, and query complexity if all parameters are fixed, except the process model, the instance part, the database, or the query, respectively.

Roadmap. As a summary of our results, Table 1 presents the complexity of the possible variants of query stability. Each section of the sequel will cover one row.

Datalog Notation. We assume familiarity with Datalog concepts such as *least fixpoint* and *stable model* semantics, and *query answering* over Datalog programs under both semantics. We consider Datalog programs that are *recursive*, *non-recursive*, *positive*, *semipositive*, *with negation*, or *with stratified negation* [9]. We write $\Pi \cup \mathcal{D}$ to denote a program where Π is a set of rules and $\mathcal{D}$ is a set of facts.

4 Undecidable MDBPs

With negation in execution conditions and writing rules, we can create MDBPs that simulate Turing machines (TMs). Consequently, in the general variant query stability is undecidable.

Due to lack of space we only provide an intuition. To show undecidability in data complexity, we define a database schema that allows us to store a TM and we construct a process model that simulates the executions of the stored TM. MDBPs cannot update facts in the database. However, we can augment relations with an additional version argument and simulate updates by adding new versions of facts. Exploiting negation in conditions and rules we can then refer to the last version of a fact. To simulate the TM execution, the process model uses fresh constants to model (i) an unbounded number of updates of the TM configurations (= number of execution steps in the TM), and (ii) a potentially infinite tape. The TM halts iff the process produces the predicate *dummy*. Undecidability in process complexity follows from undecidability in data complexity, since a process can first write the encoding of the TM into an initially empty database. Similarly, we obtain undecidability in instance complexity using instances that write the encoding of the TM at the beginning. To obtain undecidability in query complexity we extend the encoding for data complexity such that the database encodes a universal TM and an input of the TM is encoded in the query.

▶ **Theorem 3** (Undecidability). *Query stability in MDBPs is undecidable in data, process and query complexity. It is also undecidable in instance complexity except for fresh variants for which it is constant. Undecidability already holds for acyclic MDBPs.*

In our reduction it is the unbounded number of fresh instances that are causing writing rules to be executed an unbounded number of times, so that neither cycles nor existing instances are contributing to undecidability. In the sequel we study MDBPs that are positive, closed, or rowo, and show that in all three variants stability is decidable.

5 Positive Closed MDBPs

In cyclic positive MDBPs, executions can be arbitrarily long. Still, in the absence of fresh instances, it is enough to consider executions of bounded length to check stability. Consider a positive MDBP $\mathcal{B} = \langle \mathcal{P}, \mathcal{C} \rangle$, possibly with cycles and disallowing fresh instances to start, with c different constants, r relations, k running instances, m transitions and a as the maximal arity of a relation in $\mathcal{P}$. We observe:

(i) For each relation R in $\mathcal{P}$ there are up to $c^{arity(R)}$ new R-facts that $\mathcal{B}$ can produce. Thus, $\mathcal{B}$ can produce up to rc^a new facts in total.

(ii) It is sufficient to consider executions that produce at least one new fact each mk steps. An execution that produces no new facts in mk steps has at least one instance that in those mk steps visits the same place twice without producing a new fact; those steps can be canceled without affecting the facts that are produced.

(iii) Hence, it is sufficient to consider executions of maximal length $mkrc^a$.

Among these finitely many executions, it is enough to consider those that produce a maximal set of new facts. Since a process instance may have the choice among several transitions, there may be several such maximal sets. We identify a class of executions in positive closed MDBPs, called *greedy executions*, that produce all maximal sets.

Greedy Executions. Intuitively, in a greedy execution instances traverse all cycles in the net in all possible ways and produce all that can be produced before leaving the cycle. To formalize this idea we identify two kinds of execution steps: *safe steps* and *critical steps*. A safe step is an execution step of an instance after which, given the current state of the database, the instance can return to its original place. A critical step is an execution step that is not safe. Based on this, we define *greedy sequences* and *greedy executions*. A greedy sequence is a sequence of safe steps that produces the largest number of new facts possible. A greedy execution is an execution where greedy sequences and critical steps alternate.

Let Υ be a greedy execution with i alternations of greedy sequences and critical steps. In the following, we characterize which are the transitions that instances traverse in the $i+1$-th greedy sequence and then in the $i+1$-th critical step. For a process instance o and the database D_Υ produced after Υ we define the *enabled graph* $N_{\Upsilon,o}$ as the multigraph whose vertices are the places of N (i.e., the process net of $\mathcal{B}$) and edges those transitions of N that are enabled for o given database D_Υ. Let $SCC(N_{\Upsilon,o})$ denote the set of strongly connected components (SCCs) of $N_{\Upsilon,o}$. Note that two different instances may have different enabled graphs and thus different SCCs. For a place p, let $N^p_{\Upsilon,o}$ be the SCC in $SCC(N_{\Upsilon,o})$ that contains p. Suppose that o is at place p after Υ. Then in the next greedy sequence, each instance o traverses the component $N^p_{\Upsilon,o}$ in all possible ways until no new facts can be produced, meaning that all instances traverse in an arbitrary order. Conversely, the next critical step is an execution step where an instance o traverses a transition that is not part of $N^p_{\Upsilon,o}$, and thus it leaves the current SCC. We observe that when performing safe transitions new facts may be written and new transitions may become executable. This can make SCCs of $N_{\Upsilon,o}$ to grow and merge, enabling new safe steps. With slight abuse of notation we denote such maximally expanded SCCs with $N_{\Upsilon,o}$, and with $N^p_{\Upsilon,o}$ the maximal component that contains p.

Properties of Greedy Executions. We identify three main properties of greedy executions.

- A greedy execution is characterized by its critical steps, because an instance may have to choose one among several possible critical steps. In contrast, how safe steps compose a

greedy sequence is not important for stability because all greedy sequences produce the same (maximal) set of facts.

- A greedy execution in an MDBP with m transitions and k instances can have at most mk critical steps. The reason is that an execution step can be critical only the first time it is executed, and any time after that it will be a safe step.
- Each execution can be transformed into a greedy execution such that if a query is instable in the original version then it is instable also in the greedy version. In fact, an arbitrary execution has at most mk critical steps. One can construct a greedy version starting from those critical steps, such that the other steps are part of the greedy sequences.

▶ **Lemma 4.** *For each closed execution Υ in a positive MDBP $\mathcal{B}$ that produces the set of ground atoms W, there exists a greedy execution Υ' in $\mathcal{B}$ that also produces W.*

Therefore, to check stability it is enough to check stability over greedy executions. In the following we define Datalog rules that compute facts produced by greedy executions.

Encoding into Datalog. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ be a positive MDBP with m transitions and k instances. Since critical steps uniquely characterize a greedy execution, we use a tuple of size up to mk to encode them. For example, if in a greedy execution Υ at the first critical step instance o_{l_1} traverses transition t_{h_1}, in the second o_{l_2} traverses t_{h_2}, and so on up to step i, we encode this with the tuple

$$\bar{\omega} = \langle o_{l_1}, t_{h_1}, \ldots, o_{l_i}, t_{h_i} \rangle.$$

Next, we define the relations used in the encoding.

(i) For each relation R in $\mathcal{P}$ we introduce relations R^i (for i up to mk) to store all R-facts produced by an execution with i critical steps. Let Υ be the execution from above and let $\langle o_{l_1}, t_{h_1}, \ldots, o_{l_i}, t_{h_i} \rangle$ be the tuple representing it. Then, a fact of relation R^i has the form $R^i(o_{l_1}, t_{h_1}, \ldots, o_{l_i}, t_{h_i}; \bar{s})$, and it holds iff Υ produces the fact $R(\bar{s})$. Later on we use $\bar{\omega}$ to represent the tuple $\langle o_{l_1}, t_{h_1}, \ldots \rangle$. Facts of R^i are then represented as $R^i(\bar{\omega}; \bar{s})$. For convenience, we use a semicolon (;) instead of a comma (,) to separate encodings of different types in the arguments.

(ii) To record the positions of instances after each critical step we introduce relations $State^i$ such that $State^i(\bar{\omega}; p_1, \ldots, p_k)$ encodes that after Υ is executed, instance o_1 is at p_1, o_2 is at p_2, and so on.

(iii) To store the SCCs of the enabled graph we introduce relations SCC^i such that for a process instance o and a place p, the transition t belongs to $N^p_{\Upsilon,o}$ iff $SCC^i(\bar{\omega}; o, p, t)$ is true.

(iv) To compute the relations SCC^i, we first need to compute which places are reachable by an instance o from place p. For that we introduce auxiliary relations $Reach^i$ such that in the enabled graph $N_{\Upsilon,o}$ instance o can reach place p' from p iff $Reach^i(\bar{\omega}; o, p, p')$ is true.

(v) Additionally, we introduce the auxiliary relation In^0 that associates instances with their In-records, that is $In^0(o; \bar{s})$ is true iff the instance o has input record $In(\bar{s})$. With slight abuse of notation, we use $\bar{\omega}$ to denote also the corresponding greedy closed execution Υ.

In the following we define a Datalog program that computes the predicates introduced above for all possible greedy executions. The program uses stratified negation.

Initialization. For each relation R in $\mathcal{P}$ we introduce the *initialization rule* $R^0(\overline{X}) \leftarrow R(\overline{X})$ to store what holds before any critical step is made. Then we add the fact rule $State^0(p_1, \ldots, p_k) \leftarrow \mathit{true}$ if in the initial configuration o_1 is at place p_1, o_2 at p_2, and so on.

Greedy Sequence: Traversal Rules. Next, we introduce rules that compute enabled graphs. The relation $Reach^i$ contains the transitive closure of the enabled graph $N_{\bar{\omega},o}$ for each o and $\bar{\omega}$, encoding a greedy execution of length i. First, a transition t from q to be p gives rise to an edge in the enabled graph $N_{\bar{\omega},o}$ if instance o can traverse that t:

$$Reach^i(\overline{W}; O, q, p) \leftarrow E_t^i(\overline{W}; O).$$

Here, $E_t^i(\overline{W}; O)$ is a shorthand for the condition obtained from E_t by replacing $In(\bar{s})$ with $In^0(O; \bar{s})$ and by replacing each atom $R(\bar{v})$ with $R^i(\overline{W}; \bar{v})$. The tuple $\overline{W}$ consists of $2i$ many distinct variables to match every critical execution with i steps. It ensures that only facts produced by $\overline{W}$ are considered. The transitive closure is computed with the following rule:

$$Reach^i(\overline{W}; O, P_1, P_3) \leftarrow Reach^i(\overline{W}; O, P_1, P_2), Reach^i(\overline{W}; O, P_2, P_3).$$

Based on $Reach^i$, SCC^i is computed by including every transition t from q to p that an instance can reach, traverse, and from where it can return to the current place:

$$SCC^i(\overline{W}; O, P, t) \leftarrow Reach^i(\overline{W}; O, P, q), E_t^i(\overline{W}; O), Reach^i(\overline{W}; O, p, P).$$

Critical Steps: Traversal Rules. We now want to record how an instance makes a critical step. An instance o_j can traverse transition t from q to p at the critical step $i+1$ if

(i) o_j is at some place in $N^q_{\bar{\omega},o_j}$ at step i,
(ii) it satisfies the execution condition E_t,
(iii) and by traversing t it leaves the current SCC.

The following *traversal rule* captures this:

$$State^{i+1}(\overline{W}, o_j, t; P_1, \ldots, P_{j-1}, p, P_{j+1}, \ldots, P_k) \leftarrow$$
$$State^i(\overline{W}; P_1, \ldots, P_{j-1}, P, P_{j+1}, \ldots, P_k), Reach^i(\overline{W}; o_j, P, q), Reach^i(\overline{W}; o_j, q, P), \quad (1)$$
$$E_t^i(\overline{W}; o_j), \neg SCC^i(\overline{W}; o_j, P, t). \quad (2)$$

Here, the condition (i) is encoded in line (1), and (ii) and (iii) are encoded in line (2).

Generation Rules. A fact in R^{i+1} may hold because

(i) it has been produced by the current greedy sequence or by the last critical step, or
(ii) by some of the previous sequences or steps.

Facts produced by previous sequences or steps are propagated with the *copy rule*: $R^{i+1}(\overline{W}, O, T; \overline{X}) \leftarrow State^{i+1}(\overline{W}, O, T; \_), R^i(\overline{W}; \overline{X})$, copying facts $R(\overline{X})$ holding after $\overline{W}$ to all extensions of $\overline{W}$.

Then we compute the facts produced by the next greedy sequence. For each instance o_j, being at some place p_j after the last critical step in $\bar{\omega}$, and for each transition t that is in $N^{p_j}_{\bar{\omega},o_j}$, with writing rule $R(\bar{u}) \leftarrow B_t(\bar{u})$, we introduce the following *greedy generation rule*:

$$R^i(\overline{W}; \bar{u}) \leftarrow State^i(\overline{W}; \_, \ldots, \_, P_j, \_, \ldots, \_), SCC^i(\overline{W}; o_j, P_j, t), B_t^i(\overline{W}; o_j; \bar{u}),$$

where condition $B_t^i(\overline{W}; O; \bar{u})$ is obtained similarly as $E_t^i(\overline{W}; O)$. In other words, all transitions t that are in $N^{p_j}_{\bar{\omega},o_j}$ are fired simultaneously, and this is done for all instances.

The facts produced at the next critical step by traversing t, which has the writing rule $R(\bar{u}) \leftarrow B_t(\bar{u})$, are generated with the *critical generation rule*: $R^{i+1}(\overline{W}, O, t; \bar{u}) \leftarrow State^{i+1}(\overline{W}, O, t; \_), B_t^i(\overline{W}; O; \bar{u})$.

Let $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}}$ be the program encoding the positive closed $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ as described above.

▶ **Lemma 5.** *Let $\bar{\omega}$ be a greedy execution in the positive closed $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ of length i and $R(\bar{s})$ be a fact. Then $R(\bar{s})$ is produced by $\bar{\omega}$ iff $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \models R^i(\bar{\omega}; \bar{s})$.*

Test Program. We want to test the stability of $Q(\overline{X}) \leftarrow R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$. We collect all potential Q-answers using the relation Q'. A new query answer may be produced by an execution of any size i up to mk. Thus, for each execution of a size i from 0 to mk we introduce the Q'-rule

$$Q'(\overline{X}) \leftarrow R_1^i(\overline{W}; \bar{u}_1), \ldots, R_n^i(\overline{W}; \bar{u}_n). \tag{3}$$

Then, if there is a new query answer, the *test rule* "$\mathit{Instable} \leftarrow Q'(\overline{X}), \neg Q(\overline{X})$" fires the fact *Instable*. Let $\Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ be the test program that contains Q, the Q'-rules, and the test rule.

▶ **Theorem 6.** *Q is instable in the positive closed $\mathcal{B}$ iff $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \cup \Pi^{test}_{\mathcal{P},\mathcal{I},Q} \models \mathit{Instable}$.*

Data and Process Complexity. Since $\Pi^{po,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \cup \Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ is a Datalog program with stratified negation, for which reasoning is as complex as for positive Datalog, we obtain as upper bounds ExpTime for process and combined complexity, and PTime for data complexity [9]. We show that these are also lower bounds, even for singleton MDBPs. This reduction can also be adapted for acyclic fresh MDBPs, which we study in Section 8.

▶ **Lemma 7.** *Stability is* ExpTime*-hard in process and* PTime*-hard in data complexity for*

(a) *positive singleton MDBPs under closed executions, and*
(b) *positive acyclic fresh MDBPs.*

Proof Sketch.

(a) We encode query answering over a Datalog program $\Pi \cup \mathcal{D}$ into stability checking. Let A be a fact. We construct a positive singleton MDBP $\langle \mathcal{P}^{po,cl}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$, where there is a transition for each rule and the single process cycles to produce the least fixed point (LFP) of the program. In addition, the MDBP inserts the fact *dummy* if A is in the LFP. Then test query $Q_{test} \leftarrow \mathit{dummy}$ is stable in $\langle \mathcal{P}^{po,cl}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$ iff $\Pi \cup \mathcal{D} \not\models A$.
(b) Analogous, letting fresh instances play the role of the cycling singleton instance.

◀

Instance Complexity. Instance complexity turns out to be higher than data complexity. Already for acyclic positive closed MDBPs it is co-NP-hard because

(i) process instances may non-deterministically choose a transition, which creates exponentially many combinations, even in the acyclic variant; and
(ii) instances may interact by reading data written by other instances.

▶ **Lemma 8.** *There exist a positive acyclic process model $\mathcal{P}_0$, a database $\mathcal{D}_0$, and a test query Q_{test} with the following property: for every graph G one can construct an instance part $\mathcal{I}_G$ such that G is not 3-colorable iff Q_{test} is stable in $\langle \mathcal{P}_0, \mathcal{I}_G, \mathcal{D}_0 \rangle$ under closed executions.*

Clearly, Lemma 8 implies that checking stability for closed MDBPs is co-NP-hard in instance complexity. According to Theorem 11 (Section 6), instance complexity is co-NP for all closed MDBPs, which implies co-NP-completeness even for the acyclic variant.

Query Complexity. To analyze query complexity we first show how difficult it is to check whether a query returns the same answer over a database and an extension of that database.

▶ **Lemma 9** (Answer Difference)**.** *For every two fixed databases $\mathcal{D} \subseteq \mathcal{D}'$, checking whether a given conjunctive query Q satisfies $Q(\mathcal{D}) = Q(\mathcal{D}')$ is in Π^{P}_2 in the query size. Conversely, there exist databases $\mathcal{D}_0 \subseteq \mathcal{D}'_0$ such that checking for a conjunctive query Q whether $Q(\mathcal{D}_0) = Q(\mathcal{D}'_0)$ is Π^{P}_2-hard in the query size.*

Proof Idea. The first claim holds since one can check $Q(\mathcal{D}) \subsetneq Q(\mathcal{D}')$ in NP using an NP oracle. We show the second by reducing the *3-coloring extension* problem for graphs [2]. ◀

Building upon Lemma 9, we can define an MDBP that starting from $\mathcal{D}_0$ produces $\mathcal{D}'_0$. In fact, for such an MDBP it is enough to consider the simplest variants of rowo.

▶ **Proposition 10.** *Checking stability is* Π^{P}_2*-hard for*
(a) *positive fresh acyclic rowo MDBPs, and*
(b) *positive closed acyclic rowo singleton MDBPs.*

Given $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$, there are finitely many maximal extensions $\mathcal{D}'$ of $\mathcal{D}$ that can be produced by $\mathcal{B}$. We can check stability of a query Q by finitely many checks whether $Q(\mathcal{D}) = Q(\mathcal{D}')$. Since each such check is in Π^{P}_2, according to Lemma 9, the entire check is in Π^{P}_2. Thus, stability is Π^{P}_2-complete in query complexity.

6 Closed MDBPs

In the presence of negation, inserting new facts may disable transitions. During an execution, a transition may switch many times between being enabled and disabled, and greedy executions could have exponentially many critical steps. An encoding along the ideas of Section 5 would lead to a program of exponential size. This would give us an upper bound of double exponential time for combined complexity. Instead, we establish a correspondence between stability and brave query answering for Datalog with (unstratified) negation under *stable model semantics* (SMS) [9]. Due to lack of space we only state the results.

▶ **Theorem 11.** *For every closed MDBP* $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ *and every query* Q *one can construct a Datalog program with negation* $\Pi^{cl}_{\mathcal{P}}$*, based on* $\mathcal{P}$*, a database* $\mathcal{D}_{\mathcal{I}}$*, based on* $\mathcal{D}$ *and* $\mathcal{I}$*, and a test program* Π^{test}_{Q}*, based on* Q*, such that the following holds:*

$$Q \text{ is instable in } \mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle \quad \text{iff} \quad \Pi^{cl}_{\mathcal{P}} \cup \mathcal{D}_{\mathcal{I}} \cup \Pi^{test}_{Q} \models_{brave} \text{Instable}.$$

Proof Idea. For the same reason as in the positive variant, it is sufficient to consider executions of maximal length $mkrc^a$. Program $\Pi^{cl}_{\mathcal{P}}$ contains two parts:
(i) a program that generates a linear order of size $mkrc^a$ (with parameters m, k, r, c, a defined as in Section 5), starting from an exponentially smaller order, that is used to enumerate execution steps, and
(ii) a program that "guesses" an execution of size up to $mkrc^a$ by selecting for each execution step one instance and one transition, and that produces the facts that would be produced by the guessed execution. Then each execution corresponds to one stable model. The test program Π^{test}_{Q} checks if any of the guessed executions yields a new query answer. ◀

In Theorem 11, the process is encoded in the program rules while data and instances are encoded as facts. Since brave reasoning under SMS is NExpTime in program size and NP in data size [9], we have that process and combined complexity are in co-NExpTime, and data and instance complexity are in co-NP. From this and Lemma 8 it follows that instance complexity is co-NP-complete. To show that stability is co-NExpTime-complete in process and co-NP-complete in data complexity we encode brave reasoning into stability. Query complexity is Π^{P}_2-complete for the same reasons as in the positive variant.

▶ **Theorem 12.** *For every Datalog program* $\Pi \cup \mathcal{D}$*, possibly with negation, and fact* A*, one can construct a singleton MDBP* $\langle \mathcal{P}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$ *such that for the query* $Q_{test} \leftarrow$ *dummy we have:* $\Pi \cup D \models_{brave} A$ *iff* Q_{test} *is stable in* $\langle \mathcal{P}_{\Pi,A}, \mathcal{I}_0, \mathcal{D} \rangle$ *under closed executions.*

7 Acyclic Closed MDBPs

If a process net is cycle-free, all closed executions have finite length. More specifically, in an acyclic MDBP with m transitions and k running instances, the maximal length of an execution is mk. Based on this observation, we modify the encoding for the positive closed variant in Section 5 so that it can cope with negation and exploit the absence of cycles.

For an acyclic MDBP, there cannot exist any greedy steps, which would stay in a strongly connected component of the net. Therefore, we drop the encodings of greedy traversals and the greedy generation rules. We keep the rules for critical steps, but drop the atoms of relations $Reach^i$ and SCC^i. In contrast to the positive closed variant, we may have negation in the conditions E_t and B_t. However, the modified Datalog program is non-recursive, since each relation R^i and $State^i$ is defined in terms of R^j's and $State^j$'s where $j < i$.

Let $\Pi^{ac,cl}_{\mathcal{P},\mathcal{I}}$ be the program encoding an acyclic $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ as described above and let $\Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ be the test program as in the cyclic variant.

▶ **Theorem 13.** *Q is instable in the closed acyclic $\mathcal{B}$ iff* $\Pi^{ac,cl}_{\mathcal{P},\mathcal{I}} \cup \mathcal{D} \cup \Pi^{test}_{\mathcal{P},\mathcal{I},Q} \models \mathit{Instable}$.

Complexity. As upper bounds for combined and data complexity, the encoding gives us the analogous bounds for non-recursive Datalog$^\neg$ programs, that is, PSPACE in combined and AC^0 in data complexity [9]. Already in the positive variant, we inherit PSPACE-hardness of process complexity (and therefore also of combined complexity) from the program complexity of non-recursive Datalog. We obtain matching lower bounds by a reverse encoding.

▶ **Lemma 14.** *For every non-recursive Datalog program Π and every fact A, one can construct a singleton acyclic positive MDBP $\langle \mathcal{P}_{\Pi,A}, \mathcal{C}_0 \rangle$ such that for the query $Q_{test} \leftarrow dummy$ we have: $\Pi \not\models A$ iff Q_{test} is stable in $\langle \mathcal{P}_{\Pi,A}, \mathcal{C}_0 \rangle$ under closed executions.*

We observe that for closed executions, the cycles increase the complexity, and moreover, cause a split between variants with and without negation. Lemma 8 and Theorem 11 together imply that instance complexity is CO-NP-complete. Query complexity is Π^P_2-complete for the same reasons as in other closed variants.

8 Positive Fresh MDBPs

All decidable variants of MDBPs that we investigated until now were so because we allowed only closed executions. In this and the next section we show that decidability can also be guaranteed if conditions and rules are positive, or if relations are divided into read and write relations (rowo). We look first at the case where initially there are no running instances.

When fresh instances start, their input can bring an arbitrary number of new constants into the database. Thus, processes can produce arbitrarily many new facts. First we show how infinitely many executions of a positive or rowo MDBP can be faithfully abstracted to finitely many over a simplified process such that a query is stable over the original process iff it is stable over the simplified one. For such simplified positive MDBPs, we show how to encode stability checking into query answering in Datalog.

Abstraction Principle. Let $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D}, \tau_{\mathcal{B}} \rangle$ be a positive or rowo MDBP and let Q be a query that we want to check for stability. Based on $\mathcal{B}$ and Q we construct an MDBP $\mathcal{B}' = \langle \mathcal{P}', \mathcal{I}, \mathcal{D}, \tau_{\mathcal{B}} \rangle$ that has the same impact on the stability of Q but uses at most linearly many fresh values from the domain.

Let *adom* be the active domain of $\mathcal{B}$ and Q, that is the set of all constants appearing in $\mathcal{B}$ and Q. Let $\tau_1, \ldots, \tau_n$ be all timestamps including $\tau_{\mathcal{B}}$ that appear in comparisons in $\mathcal{B}$ such that $\tau_i < \tau_{i+1}$. We introduce $n+1$ many fresh timestamps $\tau'_0, \ldots, \tau'_n \notin adom$ such that $\tau'_0 < \tau_1 < \tau'_1 < \cdots < \tau_n < \tau'_n$. If there are no comparisons in $\mathcal{B}$ we introduce one fresh timestamp τ'_0. Further, let a be a fresh value such that $a \notin adom$. Let $adom^* = adom \cup \{\tau'_0, \ldots, \tau'_n\} \cup \{a\}$ be the *extended active domain*.

Then, we introduce the *discretization* function $\delta_{\mathcal{B}} \colon dom_{\mathbb{Q}^+} \to dom_{\mathbb{Q}^+}$ that based on $adom^*$ "discretizes" $dom_{\mathbb{Q}^+}$ as follows: for each $\tau \in \mathbb{Q}^+$

(i) $\delta_{\mathcal{B}}(\tau) = \tau$ if $\tau = \tau_i$ for some i;
(ii) $\delta_{\mathcal{B}}(\tau) = \tau'_i$ if $\tau_i < \tau < \tau_{i+1}$ for some i;
(iii) $\delta_{\mathcal{B}}(\tau) = \tau'_0$ if $\tau < \tau_1$;
(iv) and $\delta_{\mathcal{B}}(\tau) = \tau'_n$ if $\tau_n < \tau$;
(v) and for $c \in dom$ if $c \in adom^*$ then $\delta_{\mathcal{B}}(c) = c$; otherwise $\delta_{\mathcal{B}}(c) = a$.

If $\mathcal{B}$ has no comparisons then $\delta_{\mathcal{B}}(\tau) = \tau'_0$ for each τ. We extend $\delta_{\mathcal{B}}$ to all syntactic objects containing constants, including executions. Now, we define $\mathcal{P}'$ to be as $\mathcal{P}$, except that we add conditions on each outing transition from *start* such that only instances with values from $adom^*$ can traverse, and instances with the timestamps greater or equal than $\tau_{\mathcal{B}}$.

▶ **Proposition 15** (Abstraction). *Let $\Upsilon = \mathcal{C} \rightsquigarrow \mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \mathcal{C}_m$ be an execution in $\mathcal{B}$ that produces a set of facts W, and let $\Upsilon' = \delta_{\mathcal{B}}\Upsilon = \delta_{\mathcal{B}}\mathcal{C} \rightsquigarrow \delta_{\mathcal{B}}\mathcal{C}_1 \rightsquigarrow \cdots \rightsquigarrow \delta_{\mathcal{B}}\mathcal{C}_m$. Further, let Υ'' be an execution in $\mathcal{B}'$. Then the following holds:*

(a) *Υ' is an execution in $\mathcal{B}'$ that produces $\delta_{\mathcal{B}}W$;*
(b) *$Q(\mathcal{D}) \neq Q(\mathcal{D} \cup W)$ iff $Q(\mathcal{D}) \neq Q(\mathcal{D} \cup \delta_{\mathcal{B}}W)$;*
(c) *Υ'' is an execution in $\mathcal{B}$.*

In other words, each execution in $\mathcal{B}$ can be $\delta_{\mathcal{B}}$-abstracted and it will be an execution in $\mathcal{B}'$, and more importantly, an execution in $\mathcal{B}$ produces a new query answer if and only if the $\delta_{\mathcal{B}}$-abstracted version produces a new query answer in $\mathcal{B}'$.

Encoding into Datalog. Since $\mathcal{B}'$ allows only finitely many new values in fresh instances, there is a bound on the maximal extensions of $\mathcal{D}$ that can be produced. Moreover, since there is no bound on the number of fresh instances that can start, there is only a single maximal extension of $\mathcal{D}$, say $\mathcal{D}'$, that can result from $\mathcal{B}'$. We now define the program $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D}$ whose least fixpoint is exactly this $\mathcal{D}'$.

First, we introduce the relations that we use in the encoding. To record which fresh instances can reach a place p in $\mathcal{P}$, we introduce for each p a relation In_p with the same arity as In. That is, $In_p(\bar{s})$ evaluates to true in the program iff an instance with the input record $In(\bar{s})$ can reach p. As in the closed variant, we use a primed version R' for each relation R to store R-facts produced by the process.

Now we define the rules. Initially, all relevant fresh instances (those with constants from $adom^*$) sit at the *start* place. We encode this by the *introduction rule*: $In_{start}(X_1, \ldots, X_n) \leftarrow adom^*(X_1), \ldots, adom^*(X_n)$. Here, with slight abuse of notation, $adom^*$ represents a unary relation that we initially instantiate with the constants from $adom^*$. Also initially, we make a primed copy of each database fact, that is, for each relation R in $\mathcal{P}$ we define the *copy rule*: $R'(\overline{X}) \leftarrow R(\overline{X})$.

Then we encode instance traversals. For every transition t that goes from a place q to a place p, we introduce a *traversal rule* that mimics how instances having reached q move on to p, provided their input record satisfies the execution condition for t. Let $E_t = In(\bar{s}), R_1(\bar{s}_1), \ldots, R_l(\bar{s}_l), G_t$ be the execution condition for t, where G_t comprises the comparisons. We

define the condition $E'_t(\bar{s})$ as $In_q(\bar{s}), R'_1(\bar{s}_1), \ldots, R'_l(\bar{s}_l), G_t$, obtained from E_t by renaming the *In*-atom and priming all database relations. Then, the *traversal rule* for t is: $In_p(\bar{s}) \leftarrow E'_t(\bar{s})$. Here, $E'_t(\bar{s})$ is defined over the primed signature since a disabled transition may become enabled as new facts are produced.

Which facts are produced by traversing t is captured by a *generation rule*. Let W_t: $R(\bar{u}) \leftarrow B_t(\bar{u})$ be the writing rule for t, with the query $B_t(\bar{u}) \leftarrow In(\bar{s}'), R_1(\bar{s}'_1), \ldots, R_n(\bar{s}'_n), M_t$, where M_t comprises the comparisons. Define $B'_t(\bar{s}', \bar{u}) \leftarrow In_q(\bar{s}'), R'_1(\bar{s}'_1), \ldots, R'_n(\bar{s}'_n), M_t$. The corresponding generation rule is $R'(\bar{u}) \leftarrow E'_t(\bar{s}), B'_t(\bar{s}', \bar{u}), \bar{s} = \bar{s}'$, which combines the constraints on the instance record from E_t and W_t.

Let $\Pi^{po,fr}_{\mathcal{P},Q}$ be the program defined above, encoding the positive fresh $\mathcal{B}'$ obtained from $\mathcal{B}$. The program is constructed in such a way that it computes exactly the atoms that are in the maximal extension $\mathcal{D}'$ of $\mathcal{D}$ produced by $\mathcal{B}'$. Let $R'(\bar{v})$ be a fact.

▶ **Lemma 16.** *There is an execution in the positive fresh $\mathcal{B}$ producing $R(\bar{v})$ iff $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D} \models R'(\bar{v})$.*

Let Π^{test}_{Q} be defined like $\Pi^{test}_{\mathcal{P},\mathcal{I},Q}$ in Section 5, except that there is only one rule for Q', obtained from (3) by replacing R^i_j with R'_j. Then Proposition 15 and Lemma 16 imply:

▶ **Theorem 17.** *Q is instable the positive fresh $\mathcal{B}$ iff $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D} \cup \Pi^{test}_{Q} \models Instable$.*

Complexity. Since $\Pi^{po,fr}_{\mathcal{P},Q} \cup \mathcal{D} \cup \Pi^{test}_{Q}$ is a program with stratified negation, stability checking over positive fresh MDBPs is in ExpTime for process and combined complexity, and in PTime for data complexity [9]. From Lemma 7 we know that these are also lower bounds for the respective complexity measures. Query complexity is Π^{P}_2-complete as usual, and instance complexity is trivial for fresh processes.

Positive MDBPs. To reason about arbitrary positive MDBPs, we can combine the encoding for the fresh variant ($\Pi^{po,fr}_{\mathcal{P},Q}$) from this section and the one for the closed variant from Section 5 ($\Pi^{po,cl}_{\mathcal{P},\mathcal{I}}$). The main idea is that to obtain maximal extensions, each greedy execution sequence is augmented by also flooding the process with fresh instances. The complexities for the full positive variant are inherited from the closed variant.

9 Read-Only-Write-Only MDBPs

In general MDBPs, processes can perform recursive inferences by writing into relations from which they have read. It turns out that if relations are divided into read-only and write-only, the complexity of stability reasoning drops significantly.

The main simplifications in this case are that

(i) one traversal per instance and transition suffices, since no additional fact can be produced by a second traversal;

(ii) instead of analyzing entire executions, it is enough to record which paths an individual process instance can take and which facts it produces, since instances cannot influence each other.

As a consequence, the encoding program can be non-recursive and it is independent of the instances in the process configuration. A complication arises, however, since the maximal extensions of the original database $\mathcal{D}$ by the MDBP $\mathcal{B}$ are not explicitly represented by this approach. They consist of unions of maximal extensions by each instance and are encoded into the test query, which is part of the program.

▶ **Theorem 18.** *For every rowo MDBP $\mathcal{B} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$ and query Q one can construct a nonrecursive Datalog program $\Pi^{ro}_{\mathcal{P},Q}$, based on $\mathcal{P}$ and Q, and a database instance $\mathcal{D}_{\mathcal{I}}$, based on $\mathcal{D}$ and $\mathcal{I}$, such that: Q is instable in $\mathcal{B}$ iff $\Pi^{ro}_{\mathcal{P},Q} \cup \mathcal{D}_{\mathcal{I}} \models$ Instable.*

From the theorem it follows that data and instance complexities are in AC^0, except for instance complexity in fresh variants, for which it is constant.

Process, Query and Combined Complexity. Since CQ evaluation can be encoded into an execution condition, this gives us co-NP-hardness of stability in process complexity. We also show that it is in co-NP. First we note that due to the absence of recursion, one can check in NP whether a set of atoms is produced by a process instance.

▶ **Proposition 19.** *Let $\mathcal{B}$ be a singleton rowo MDBP. One can decide in* NP*, whether for given facts $A_1, \ldots, A_m$, there is an execution in $\mathcal{B}$ that produces $A_1, \ldots, A_m$.*

Next, suppose that $\mathcal{I}$, $\mathcal{D}$ and $Q(\bar{v}) \leftarrow B_1, \ldots, B_m$ are a fixed instance part, database and query. Given a process model $\mathcal{P}$, we want to check that Q is instable in $\mathcal{B}_{\mathcal{P}} = \langle \mathcal{P}, \mathcal{I}, \mathcal{D} \rangle$. Making use of the abstraction principle for fresh constants, we can guess in polynomial time an instantiation $B'_1, \ldots, B'_n$ of the body of Q that returns an answer not in $Q(\mathcal{D})$. Then we verify that $B'_1, \ldots, B'_n$ are produced by $\mathcal{B}_{\mathcal{P}}$. Such a verification is possible in NP according to Proposition 19. We guess a partition of the set of facts $B'_1, \ldots, B'_n$, guess one instance, possibly fresh, for each component set of the partition, and verify that the component set is produced by the instance. Since all verification steps were in NP, the whole check is in NP.

Query complexity is Π^{P}_2-complete for the same reasons as in the general variant, and one can show that this is also the upper-bound for the combined complexity.

10 Related Work

Traditional approaches for business process modeling focus on the set of activities to be performed and the flow of their execution. These approaches are known as *activity-centric*. A different perspective, mainly investigated in the context of databases, consists in identifying the set of data (entities) to be represented and describes processes in terms of their possible evolutions. These approaches are known as *data-centric*.

In the context of activity-centric processes, Petri Nets (PNs) have been used for the representation, validation and verification of formal properties, such as absence of deadlock, boundedness and reachability [26, 27]. In PNs and their variants, a token carries a limited amount of information, which can be represented by associating to the token a set of variables, like in colored PNs [18]. No database is considered in PNs.

Among data-centric approaches, *Transducers* [1, 25] were among the first formalisms ascribing a central role to the data and how they are manipulated. These have been extended to *data driven web systems* [11] to model the interaction of a user with a web site, which are then extended in [10]. These frameworks express insertion and deletion rules using FO formulas. The authors verify properties expressed as FO variants of LTL, CTL and CTL* temporal formulas. The verification of these formulas results to be undecidable in the general case. Decidability is obtained under certain restrictions on the input, yielding to ExpSpace complexity for checking LTL formulas and co-NExpTime and ExpSpace for CTL and CTL* resp., in the propositional case.

Data-Centric Dynamic Systems (DCDSs) [4] describe processes in terms of guarded FO rules that evolve the database. The authors study the verification of temporal properties

expressed in variants of μ-calculus (that subsumes CTL*-FO). They identify several undecidable classes and isolate decidable variants by assuming a bound on the size of the database at each step or a bound on the number of constants at each run. In these cases verification is ExpTime-complete in data complexity.

Overall, both frameworks are more general than MDBPs, since deletions and updates of facts are also allowed. This is done by rebuilding the database after each execution step. Further, our stability problem can be encoded as FO-CTL formula. However, our decidability results for positive MDBPs are not captured by the decidable fragments of those approaches. In addition, the authors of the work above investigate the borders of decidability, while we focus on a simpler problem and study the sources of complexity. Concerning the process representation, both approaches adopt a rule-based specification. This makes the control flow more difficult to grasp, in contrast to activity-centric approaches where the control flow has an explicit representation.

Artifact-centric approaches [17] use artifacts to model business relevant entities. In [6, 14, 15] the authors investigate the verification of properties of artifact-based processes such as reachability, temporal constraints, and the existence of dead-end paths. However, none of these approaches explicitly models an underlying database. Also, the authors focus on finding suitable restrictions to achieve decidability, without a fine-grained complexity analysis as in our case.

Approaches in [3] and [5], investigate the challenge of combining processes and data, however, focusing on the problem of data provenance and of querying the process structure.

In [12, 20] the authors study the problem of determining if a query over views is independent from a set of updates over the database. The authors do not consider a database instance nor a process. Decidability in rowo MDBPs can be seen as a special case of those.

In summary, our approach to process modeling is closer to the activity-centric one but we model manipulation of data like in the data-centric approaches. Also, having process instances and MDBPs restrictions gives finer granularity compared to data-centric approaches.

11 Discussion and Conclusion

Discussion. An interesting question is how complex stability becomes if MDBPs are not monotonic, i.e., if updates or deletions are allowed. In particular, for positive MDBPs we can show the following. In acyclic positive closed MDBPs updates and deletions can be modeled using negation in the rules, thus stability stays PSpace-complete. For the cyclic positive closed variant, allowing updates or deletions is more powerful than allowing negation, and stability jumps to ExpSpace-completeness. For positive MDBPs with updates or deletions stability is undecidable.

In case the initial database is not known, our techniques can be still applied since an arbitrary database can be produced by fresh instances starting from an empty database.

Contributions. Reasoning about data and processes can be relevant in decision support to understand how processes affect query answers.

1. To model processes that manipulate data we adopt an explicit representation of the control flow as in standard BP languages (e.g., BPMN). We specify how data is manipulated as annotations on top of the control flow.
2. Our reasoning on stability can be offered as a reasoning service on top of the query answering that reports on the reliability of an answer. Ideally, reasoning on stability should not bring a significant overhead on query answering in practical scenarios. Existing

work on processes and data [4] shows that verification of general temporal properties is typically intractable already measured in the size of the data.

3. In order to identify tractable cases and sources of complexity we investigated different variants of our problem, by considering negation in conditions, cyclic executions, read access to written data, presence of pending process instances, and the possibility to start fresh process instances.
4. Our aim is to deploy reasoning on stability to existing query answering platforms such as SQL and ASP [19]. For this reason we established different encodings into suitable variants of Datalog, that are needed to capture the different characteristics of the problem. For each of them we showed that our encoding is optimal. In contrast to existing approaches, which rely on model checking to verify properties, in our work we rely on established database query languages.

Open Questions. In our present framework we cannot yet model process instances with activities that are running in parallel. Currently, we are able to deal with it only in case instances do not interact (like in rowo). Also, we do not know yet how to reason about expressive queries, such as conjunctive queries with negated atoms, and first-order queries. From an application point of view, stability of aggregate queries and aggregates in the process rules are relevant. A further question is how to quantify instability, that is, in case a query is not stable, how to compute the minimal/maximal number of possible new answers.

Acknowledgments. This work was partially supported by the research projects MAGIC, funded by the province of Bozen-Bolzano, and CANDy and PARCIS, funded by the Free University of Bozen-Bolzano.

References

1 S. Abiteboul, V. Vianu, B.S. Fordham, and Y. Yesha. Relational Transducers for Electronic Commerce. In *PODS*, pages 179–187, 1998. doi:10.1145/275487.275507.

2 M. Ajtai, R. Fagin, and L.J. Stockmeyer. The Closure of Monadic NP. *J. Comput. Syst. Sci.*, 60(3):660–716, 2000.

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 B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of Relational Data-Centric Dynamic Systems with External Services. In *PODS*, pages 163–174, 2013. doi:10.1145/2463664.2465221.

5 C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying Business Processes. In *VLDB*, pages 343–354, 2006.

6 K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *BPM*, pages 288–304, 2007.

7 Bonitasoft. Bonita BPM. Accessed: 2015-12-16. URL: http://www.bonitasoft.com.

8 G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving Data Quality: Consistency and Accuracy. In *VLDB*, pages 315–326, 2007.

9 E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

10 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic Verification of Data-Centric Business Processes. In *ICDT*, pages 252–267, 2009. doi:10.1145/1514894.1514924.

11 A. Deutsch, L. Sui, and V. Vianu. Specification and Verification of Data-Driven Web Services. In *PODS*, pages 71–82, 2004.

12 C. Elkan. Independence of Logic Database Queries and Updates. In *PODS*, pages 154–160, 1990. doi:10.1145/298514.298557.

13 W. Fan, F. Geerts, and J. Wijsen. Determining the Currency of Data. *ACM Trans. Database Syst.*, 37(4):25, 2012.

14 C. E. Gerede, K. Bhattacharya, and J. Su. Static Analysis of Business Artifact-Centric Operational Models. In *SOCA*, pages 133–140, 2007.

15 C. E. Gerede and J. Jianwen Su. Specification and Verification of Artifact Behaviors in Business Process Models. In *ICSOC*, pages 181–192, 2007.

16 F. T. Heath, D. Boaz, M. Gupta, R. Vaculín, Y. Sun, R. Hull, and L. Limonad. Barcelona: A Design and Runtime Environment for Declarative Artifact-Centric BPM. In *ICSOC*, pages 705–709, 2013. doi:10.1007/978-3-642-45005-1_65.

17 R. Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In *OTM*, pages 1152–1163, 2008.

18 K. Jensen and L.M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.

19 N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

20 A.Y. Levy and Y. Sagiv. Queries Independent of Updates. In *VLDB*, pages 171–181, 1993.

21 E. Marengo, W. Nutt, and O. Savković. Towards a Theory of Query Stability in Business Processes. In *AMW*, volume 1189 of *CEUR Workshop Proceedings*, 2014.

22 Object Management Group. *Business Process Model and Notation 2.0 (BPMN)*, Jan 2011. URL: http://www.omg.org/spec/BPMN/2.0/.

23 S. Razniewski and W. Nutt. Completeness of Queries over Incomplete Databases. *PVLDB*, 4(11):749–760, 2011.

24 O. Savković, E. Marengo, and W. Nutt. Query Stability in Data-aware Business Processes. Technical Report KRDB15-1, KRDB Research Center, Free Univ. Bozen-Bolzano, 2015. URL: http://www.inf.unibz.it/krdb/pub/tech-rep.php.

25 M. Spielmann. Verification of Relational Transducers for Electronic Commerce. In *PODS*, pages 92–103. ACM, 2000.

26 W.M.P. van der Aalst. Verification of Workflow Nets. In *ICATPN*, pages 407–426, 1997. doi:10.1007/3-540-63139-9_48.

27 W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.