The Data Readiness Problem for Relational Databases

Rada Chirkova[1], Jon Doyle[1], and Juan L. Reutter[2]

[1] North Carolina State University
[2] Pontificia Universidad Católica de Chile

Abstract. We consider the problem of determining whether organizations facing a new data-transformation task can avoid building a new transformation procedure from scratch by reusing their stored procedures. Because it can be difficult to obtain exact descriptions of what stored procedures do, our framework abstracts data-transforming tools as black-box procedures, in which a procedure description indicates the parts of the database that might be modified by the procedure and constraints on the states of the database that must hold before and after the application of this procedure.
In this paper we present our framework and study the problem of determining, given a database and a set of procedures, whether there is a sequence of procedures from this set such that their application to the database results in the satisfaction of a boolean query. This *data readiness* problem is undecidable in general, but we show decidability for a broad and realistic class of procedures.

1 Introduction

The databases of many organizations nowadays periodically undergo transformations, due to applications of data-improvement operations, merging of multiple repositories, or management decisions. These transformations are commonly carried out by means of stored procedures or other similar artifacts that are kept together with the database, and may have to be applied periodically, becoming at times part of the normal daily operations of the organizations. For example, it is not uncommon to find institutions with separate databases for their accountancy and operations divisions, in which the integration is carried out by a stored procedure that runs at the end of every working day.

Whenever a new data-transformation task arises, organizations facing the cost of assembling a new procedure to solve this task may ask instead whether one can reuse some of the procedures that are already available. However, to answer this question we need to be able to reason about the outcomes of procedures, or even of sequences of applications of procedures. Several lines of research have been studying these outcomes when procedures are understood as part of the normal operations of an institution (see excellent surveys [14, 8, 1]); many of these works assume a complete description of all the procedures involved in these operations.

At the same time, it is not always feasible to obtain an exact description of the inner workings of a procedure (as in when its creator(s) no longer work for the company, see, e.g., [13]). In such cases, one has to work with informal or vague descriptions of what procedures do: "This procedure copies relation A into relation B," or "this procedure removes all nulls from this relation." To model such uncertainty, we do not assume that we have a precise description of the operations of the procedures, and adopt instead a black-box view of a procedure. That is, we describe procedures in terms of which parts of the data might be modified by the procedure, as well as by the constraints that specify the required states of the data before and after applying the procedure.

Motivating example: Suppose a medical analyst wishes to know the emergency rooms used by patients with a certain medical insurance. The data owned by the analyst reside in relation *LocVisits* (*facility*,*pId*,*timestp*), with the attributes standing, respectively, for the id of the facility where the emergency room is, the social-security number of a patient, and a timestamp marking the date of the visit. The analyst has also been given two procedures he can execute as-is but not modify: One is P_{migrate}, which is supposed to migrate data into *LocVisits* from relation *EVisits* owned by another analysis company. The other procedure, P_{insur}, augments *LocVisits* with an attribute *insId* containing the insurance id's of patients, and whose data are drawn from relation *Patients*(*pId*, *insId*) owned by the local authority.

Given an insurance id I, the analyst can capture the desired information via query `SELECT facility FROM LocVisits WHERE insId = I`, posed over *LocVisits* modified by adding attribute *insId* containing the insurance id's of patients. It is natural for the analyst to ask: Can I use any available procedures to transform my data so that this query can be posed on my database? In other words, is there a way to apply these procedures so that I could guarantee that my database satisfies certain fitness-for-use [12] criteria?

We propose a formal framework in which data-transforming tools are abstracted as black-box procedures, described by the following information:

- A specification of which parts of the database the procedure is modifying;
- Conditions to be satisfied for the procedure to be applicable;
- Conditions that will be satisfied once the procedure has been applied; and
- Any additional guarantees on parts of the data that must not be modified.

In this paper we study this framework for procedures that do not alter the schema of databases, such as the procedure that migrates the information of the analyst in the example above. We study basic questions arising in the framework, such as whether a procedure can be applied to the outcome of a given procedure over a given instance, and whether the outcome of a (sequence of) procedures is nonempty. Finally, we consider what we call the *data-readiness problem:* Given an instance I, a set Π of procedures, and a boolean query over instances (that intuitively expresses a desired property of the data), is there a way to construct a sequence of procedures from Π so that each instance in the outcome satisfies this property? While undecidable in its general form, we show that this problem is decidable for some broad classes of procedures.

For space reasons we omit proofs from this draft. All of them can be found in the full version of this paper [5].

2 Preliminaries

Since we aim to model procedures over real databases, we write the paper using a specific named assumption over instances and queries.

Schemas and Instances. Assume three disjoint sets: a countably infinite set of attribute names $\mathcal{A} = \{A_1, A_2, \ldots\}$ totally ordered by $\leq_{\mathcal{A}}$, a countably infinite domain of values (or elements) D, and a countably infinite set of relation names $\mathcal{R} = \{R_1, R_2, \ldots\}$. A relational schema over $\mathcal{A}$ and $\mathcal{R}$ is a partial function $\mathcal{S} : \mathcal{R} \to 2^{\mathcal{A}}$, which associates a finite set of attributes with a finite set of relation symbols. We say that R is in $\mathcal{S}$ if $\mathcal{S}(R)$ is defined. An instance I of schema $\mathcal{S}$ assigns a set R^I of tuples to each relation R in $\mathcal{S}$, so that if $\mathcal{S}(R) = \{A_1, \ldots, A_n\}$ then $R^I \subseteq D^n$, with the set of tuples structured so that the elements of each tuple $(a_1, \ldots, a_n)$ appear in the assumed attribute order, that is, $A_1 <_{\mathcal{A}} \cdots <_{\mathcal{A}} A_n$.

Queries across Schemas, Total Queries. A *named atom* is an expression $R(A_1 : x_1, \ldots, A_k : x_k)$, where R is a relation name, each A_i is an attribute name, and each x_i is a variable. The variables mentioned by such an atom are $x_1, \ldots, x_k$, and the attributes are $A_1, \ldots, A_k$. Such an atom is *compatible* with schema $\mathcal{S}$ if $\{A_1, \ldots, A_k\} \subseteq \mathcal{S}(R)$. Given a named atom $R(A_1 : x_1, \ldots, A_k : x_k)$, an instance I of schema $\mathcal{S}$ that is compatible with the atom, and an assignment $\tau : \{x_1, \ldots, x_k\} \to D$ assigning values to variables, we say that (I, τ) *satisfies* $R(A_1 : x_1, \ldots, A_k : x_k)$ if there is a tuple a in R^I such that its projection $\pi_{A_1,\ldots,A_k}\ \bar{a}$ over $A_1, \ldots, A_k$ is the tuple $\tau(x_1), \ldots, \tau(x_k)$.

A *conjunctive query* (CQ) is an expression of the form $\exists \bar{z}\ \phi(\bar{z}, \bar{y})$, where $\bar{z}$ and $\bar{y}$ are tuples of variables and $\phi(\bar{z}, \bar{y})$ is a conjunction of named atoms that uses the variables in $\bar{z}$ and $\bar{y}$. A CQ is compatible with $\mathcal{S}$ if all its named atoms are compatible. The usual semantics of CQs is obtained from the semantics of named atoms in the usual way. Given a CQ Q compatible with $\mathcal{S}$, the result $Q(I)$ of evaluating Q over I is the set of all the tuples $\tau(x_1), \ldots, \tau(x_k)$ such that (I, τ) satisfy Q. Further, a *total query*, which we define to be an expression of the form R for some relation name R, returns all the tuples of R, regardless of the schema and arity of R, as is done in SQL with `SELECT * FROM R`. A total query of this form is compatible with schema $\mathcal{S}$ if $\mathcal{S}(R)$ is defined; the result of evaluating this query over an instance I over a compatible schema $\mathcal{S}$ is the set of all tuples in R^I.

Data constraints: We consider data constraints that are (i) *tuple-generating dependencies (tgds)*, i.e., expressions of the form $\forall \bar{x}\big(\exists \bar{y}\ \phi(\bar{x}, \bar{y}) \to \exists \bar{z}\ \psi(\bar{x}, \bar{z})\big)$ for CQs $\exists \bar{y}\ \phi(\bar{x}, \bar{y})$ and $\exists \bar{z}\ \psi(\bar{x}, \bar{z})$, and (ii) *equality-generating dependencies (egds)*, i.e., expressions of the form $\forall \bar{x}\big(\exists \bar{y}\ \phi(\bar{x}, \bar{y}) \to x = x'\big)$ for a CQ $\exists \bar{y} \phi(\bar{x}, \bar{y})$ and variables x, x' in $\bar{x}$. As usual, for readability we sometimes omit the universal quantifiers of tgds and egds. An instance I *satisfies* a set Σ of tgds and egds, written $I \models \Sigma$, if (1) each CQ in each dependency in Σ is compatible with the

schema of I, and (2) every assignment $\tau : \bar{x} \cup \bar{y} \to D$ such that $(I, \tau) \models \phi(\bar{x}, \bar{y})$ can be extended into a $\tau' : \bar{x} \cup \bar{y} \cup \bar{z} \to D$ such that $(I, \tau') \models \psi(\bar{x}, \bar{z})$.

A tgd is *full* if it does not use existentially quantified variables on the right-hand side. A set Σ of tgds is *full* if each tgd in Σ is full. Σ is *acyclic* if an acyclic graph is formed by representing each relation mentioned in a tgd in Σ as a node and by adding an edge from node R to S if a tgd in Σ mentions R on the left-hand side and S on the right-hand side.

3 Procedures

In this section we formalize the notion of procedures that transform data. We view procedures as black boxes and assume no knowledge of or control over their inner workings. Our reasoning about procedures is based on the following information: The input conditions, or *preconditions,* on the state of the data that must hold for a procedure to be applicable; the output conditions, or *post-conditions,* on the state of the data that must hold after an application of the procedure; and the set of relations affected by the application. To specify that some of the data will not be deleted, we also allow the inclusion of some queries whose answer needs to be preserved during the application of the procedure.

Example 1. Recall the procedure P_{migrate} outlined in Section 1; its intent is to define migration of data from relation *EVisits* into *LocVisits*. P_{migrate} can be described as follows. First, the *scope*: P_{migrate} only changes relation *LocVisits*. Next, the *precondition*: P_{migrate} requires a schema with relations *LocVisits* and *EVisits*, each with attributes *facility*, *pId*, and *timestp*. Next, the *postcondition*: after P_{migrate} is applied, each tuple of *EVisits* must be in *LocVisits*. Finally, we need to *guarantee* that the tuples in *LocVisits* are not deleted.

In the following, we present notation for formally defining these types of procedures. We start by introducing "structure constraints," which we use to define the scopes of procedures.

3.1 Structure Constraints

A *structure constraint* is a formula of the form $R[\bar{s}]$ or $R[*]$, with R a relation symbol, $\bar{s}$ a tuple of attribute names from $\mathcal{A}$, and $*$ a symbol not in $\mathcal{A}$ or $\mathcal{R}$ acting as a wildcard. A *schema $\mathcal{S}$ satisfies a structure constraint* $R[\bar{s}]$, denoted $\mathcal{S} \models R[\bar{s}]$, if $\mathcal{S}(R)$ is defined and each attribute in $\bar{s}$ belongs to $\mathcal{S}(R)$. A *schema $\mathcal{S}$ satisfies the constraint* $R[*]$ if $\mathcal{S}(R)$ is defined.

Given a set $\mathcal{C}$ of structure constraints and a schema $\mathcal{S}$, we denote by $Q_{\mathcal{S}\setminus\mathcal{C}}$ the conjunctive query formed by the conjunction of the following atoms:

- For each relation R such that $\mathcal{S}(R) = \{A_1, \ldots, A_m\}$ and R is not mentioned in $\mathcal{C}$, $Q_{\mathcal{S}\setminus\mathcal{C}}$ includes an atom $R(A_1 : z_1, \ldots, A_m : z_m)$, where $z_1, \ldots, z_m$ are fresh variables.

– For each T mentioned in $\mathcal{C}$ such that $T[*]$ is not in $\mathcal{C}$, $Q_{\mathcal{S}\setminus\mathcal{C}}$ includes an atom $T(B_1 : z_1, \ldots, B_k : z_k)$, where $B_1, \ldots, B_k$ are all the attributes in $\mathcal{S}(T)$ that are not mentioned in any constraint of the form $T[\bar{s}]$ in $\mathcal{C}$, and $z_1, \ldots, z_k$ are fresh variables.

Intuitively, $Q_{\mathcal{S}\setminus\mathcal{C}}$ is intended to retrieve the projection of the entire database over all the relations and attributes not mentioned in $\mathcal{C}$. $Q_{\mathcal{S}\setminus\mathcal{C}}$ is unique up to the variable renaming and order of conjuncts. As an example, let schema $\mathcal{S}$ have relations R, S, and T, with attributes: A_1 and A_2 in R; B_1, B_2, and B_3 in T; and A_1 and B_1 in S. Let set $\mathcal{C}$ comprise constraints $R[*]$ and $S[B_1]$. Then $Q_{\mathcal{S}\setminus C}$ is the query $T(B_1 : z_1, B_2 : z_2, B_3 : z_3) \wedge S(A_1 : w_1)$.

3.2 Formal Definition of Procedures

We define procedures w.r.t. a class $\mathbb{C}$ of FO constraints and a class $\mathbb{Q}$ of queries. We will focus primarily on tgds, egds, structure constraints, and CQ queries.

Definition 1. *A* procedure P *over* $\mathbb{C}$ *and* $\mathbb{Q}$ *is a tuple* $(Scope, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{pres})$, *where:*

– *Scope is a set of structure constraints that defines the scope (i.e., the relations and attributes) within which the procedure operates;*
– $\mathcal{C}_{in}$ *and* $\mathcal{C}_{out}$ *are constraints in* $\mathbb{C}$ *describing the pre- and postconditions of* P*, respectively;*
– $\mathcal{Q}_{pres}$ *is a set of queries in* $\mathbb{Q}$ *that serve as a preservation guarantee for the procedure.*

Example 2 (Example 1 continued). We define the procedure P_{migrate} formally as follows:

Scope: The scope is the constraint *LocVisits*$[*]$.

$\mathcal{C}_{\text{in}}$: We use the structure constraints *EVisits*[*facility*, *pId*, *timestp*] and *LocVisits* [*facility*, *pId*, *timestp*], to ensure that the data have the correct attributes.

$\mathcal{C}_{\text{out}}$: The postcondition comprises the tgd

$$EVisits(facility : x, pId : y, timestp : z) \rightarrow LocVisits(facility : x, pId : y, timestp : z).$$

It says that, once P_{migrate} has been applied, the projection of *EVisits* over *facility*, *pId*, and *timestp* is a subset of the respective projection of *LocVisits*.
$\mathcal{Q}_{\text{pres}}$: We use query $LocVisits(facility : x, pId : y, timestp : z)$, whose intent is to state the guarantee that all the answers on *LocVisits* that are present before P_{migrate} is applied will be preserved.

Semantics: A procedure $P = (Scope, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{pres}})$ is *applicable* on an instance I over schema $\mathcal{S}$ if (1) Each query in $\mathcal{Q}_{\text{pres}}$ is compatible with $\mathcal{S}$; and (2) $I \models \mathcal{C}_{\text{in}}$. We can now proceed with the semantics of procedures.

Definition 2. *Let* I *be an instance over schema* $\mathcal{S}$*. An instance* I' *over* $\mathcal{S}$ *is a* possible outcome *of applying procedure* P *to* I *if all of the following holds:*

1. *P is applicable on I;*
2. *$I' \models \mathcal{C}_{out}$;*
3. *The answers of the query $Q_{\mathcal{S}\setminus Scope}$ do not change: $Q_{\mathcal{S}\setminus Scope}(I) = Q_{\mathcal{S}\setminus Scope}(I')$; and*
4. *The answers to each query Q in $\mathcal{Q}_{pres}$ over I are preserved: $Q(I) \subseteq Q(I')$.*

Example 3 (Example 2 continued). Recall procedure $P_{\text{migrate}} = (Scope, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{pres}})$ defined in Example 2. Consider instance I over schema $\mathcal{S}$ with relations *EVisits* and *LocVisits*, each with attributes *facility*, *pId*, and *timestp*, as shown in Figure 1 (a). Note first that P_{migrate} is indeed applicable on I. When applying P_{migrate} to I, we know from *Scope* that the only relation whose content can change is *LocVisits*, while *EVisits* is the same across all possible outcomes. Further, we know from $\mathcal{C}_{\text{out}}$ that in all possible outcomes, the projection of *EVisits* over the attributes *facility*, *pId*, and *timestp* must be the same as the projection of *LocVisits* over the same attributes. Finally, from $\mathcal{Q}_{\text{pres}}$ we know that the projection of *LocVisits* over these three attributes must be preserved.

Perhaps the most obvious possible outcome of applying P_{migrate} to I is that of the instance J_1 in Figure 1 (b), corresponding to the outcome where the tuple in *EVisits* that was not yet in *LocVisits* is migrated into the latter relation. However, since we assume no control over the actions performed by P_{migrate}, it may well be that it is also migrating data from a different relation that we are not aware of, producing an outcome whose relation *EVisits* is the same as in I and J_1, but *LocVisits* has additional tuples, as depicted in Figure 1 (c).

As seen in Example 3, in general the number of possible outcomes that result from applying a procedure is infinite. Thus, we are in general interested in properties shared by all possible outcomes, which motivates the following definition.

Definition 3. *The* outcome set *of applying a procedure P to I is defined as:*

$$outcomes_P(I) = \{I' \mid I' \text{ is a possible outcome of applying } P \text{ to } I\}.$$

The outcome of applying a procedure P to a set of instances $\mathcal{I}$ is the union of the outcome sets of applying P to all the instances in $\mathcal{I}$:

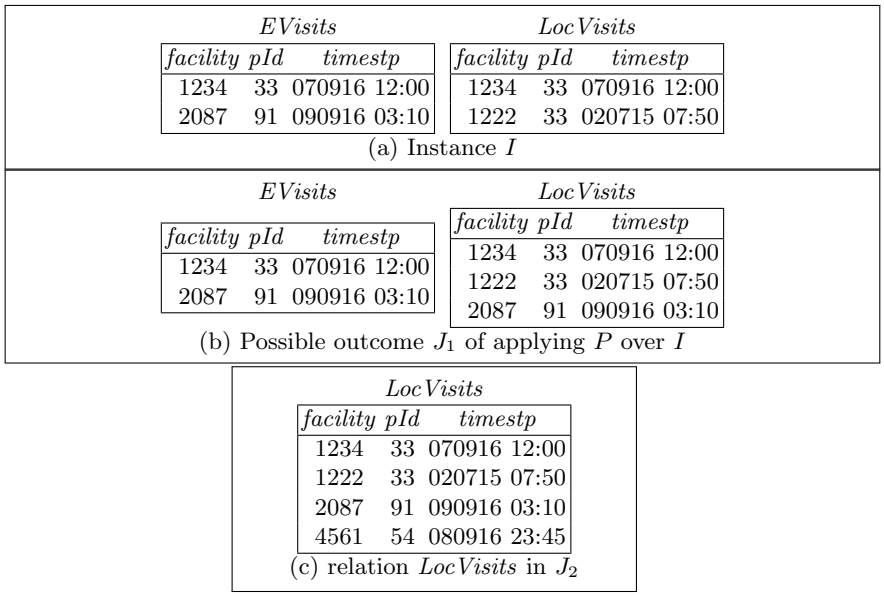
$$outcomes_P(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} outcomes_P(I).$$

Finally, to reason about (perhaps repeated) applications of multiple procedures, we extend the definitions to enable talking about the outcomes of sequences of procedures. The outcome of applying a sequence $P_1, \ldots, P_n$ of procedures to instance I is the set

$$outcomes_{P_1,\ldots,P_n}(I) = outcomes_{P_n}(outcomes_{P_{n-1}}(\cdots(outcomes_{P_1}(I))\cdots)).$$

4 Basic Decision Problems

We begin with two decision problems on outcomes of sequences of procedures.



EVisits

facility	*pId*	*timestp*
1234	33	070916 12:00
2087	91	090916 03:10

LocVisits

facility	*pId*	*timestp*
1234	33	070916 12:00
1222	33	020715 07:50

(a) Instance I

EVisits

facility	*pId*	*timestp*
1234	33	070916 12:00
2087	91	090916 03:10

LocVisits

facility	*pId*	*timestp*
1234	33	070916 12:00
1222	33	020715 07:50
2087	91	090916 03:10

(b) Possible outcome J_1 of applying P over I

LocVisits

facility	*pId*	*timestp*
1234	33	070916 12:00
1222	33	020715 07:50
2087	91	090916 03:10
4561	54	080916 23:45

(c) relation *LocVisits* in J_2

Fig. 1. Instance I of Example 3 (a), a complete possible outcome of applying P_{migrate} to I (b), and the instance of *LocVisits* in another possible outcome in which *LocVisits* has an additional tuple not mentioned in *EVisits* (c).

4.1 Applicability:

Suppose we wish to apply procedures P_1 and P_2 to instance I sequentially, first P_1 then P_2. To ensure applicability of P_2, we need to guarantee that *any* possible outcome of applying P_1 to I will satisfy the preconditions of P_2. Hence, we study the following *applicability* problem: Given schema $\mathcal{S}$ and procedures P_1 and P_2, is it true that P_2 can be applied to each instance in $outcomes_{P_1}(I)$, regardless of the choice of $I \in \mathcal{S}$?

The applicability problem is intimately related to the problem of implication of dependencies (see, e.g., [2–4, 11, 6, 7]). Indeed, consider a class $\mathcal{L}$ of constraints for which the implication problem is known to be undecidable. Then if we let P_1 be a procedure with a set Σ of postconditions in $\mathcal{L}$, and P_2 a procedure with a dependency λ in $\mathcal{L}$ as a precondition, it is not difficult to come up with proper scopes and preservation queries so that the set $outcomes_{P_1}(I)$ satisfies λ for every instance I over schema $\mathcal{S}$ if and only if λ is true in all instances that satisfy Σ.

However, applicability turns out to be undecidable even if we fix P_1 and P_2:

Proposition 1. *There are fixed procedures P_1 and P_2 that only use tgds for their constraints, and such that the following problem is undecidable: Given an instance I over schema $\mathcal{S}$, is it true that all the instances in $outcomes_{P_1}(I)$ satisfy the preconditions of P_2?*

The proof of Proposition 1 is by reduction from the embedding problem for finite semigroups, shown to be undecidable in [10].

This result suggests that to obtain decidability, it may not be enough to focus on procedures whose pre- and postconditions are specified in languages with decidable implication problem — one would need to create stronger restrictions. Consider, e.g., the restriction of disallowing preconditions in procedures, for which we have the following trivial result.

Fact 1 *A procedure without preconditions can always be applied to any instance.*

Nonemptiness: The other important problem is that of determining whether the outcome of a sequence of procedures is nonempty. We remark that even without preconditions, the outcome of a procedure may be empty if it is not possible to transform an instance in a way that would satisfy the postconditions of a procedure, while ensuring that the scope and preservation queries are respected. Perhaps surprisingly, we can show that this problem is undecidable even if we just have one fixed procedure.

Proposition 2. *There exists a procedure P that does not use preconditions and uses only tgds in its postconditions, such that the following is undecidable: Given an instance I, is the set $outcomes_P(I)$ nonempty?*

The proof of this proposition is similar to that of Proposition 1, but in this case we can get away with one procedure by merging together the postconditions and precondition of the procedures in the reduction for Proposition 1.

Procedures with safe scope: Toward obtaining decidability, we could continue restricting the types of constraints we allow in procedures. (For example, nonemptiness is decidable for a broad range of acyclicity conditions.) We choose to adopt a different strategy, which restricts the interplay between the postconditions of procedures, their scope, and their preservation queries.

We say that procedure $P = (\mathit{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{pres}})$ has *safe scope* if the following holds:

- $\mathcal{C}_{\text{in}}$ is empty, and $\mathcal{C}_{\text{out}}$ is an acyclic set of tgds;
- The set *Scope* contains exactly one constraint $R[*]$ for each relation R that appears on the right-hand side of a tgd in $\mathcal{C}_{\text{out}}$; and
- The set Q_{pres} contains one total query R for each constraint $R[*]$ in *Scope*. That is, it binds precisely all the relations in the scope of P.

Note that the procedure P_{migrate} of Example 2, while not a procedure with safe scope, can easily be transformed into one. Once again, we have an easy result that makes a case for the good behaviour of procedures with safe scope:

Proposition 3. *For every instance I and sequence $P_1, \ldots, P_n$ of procedures with safe scope, the set $outcomes_{P_1,\ldots,P_n}(I)$, is not empty.*

5 Data Readiness

We now address the problem of assessing achievability of desired properties of data, which we describe informally as follows. We start with an instance I and

have a set Π of procedures. We are also given a boolean query Q (that intuitively expresses a desired property) that does not hold in I. The question we ask is whether we can apply to I some or all the procedures in Π so that all the resulting outcomes would satisfy Q:

DATA READINESS:
Input: An instance I, a set Π of procedures, and a boolean query Q;
Question: Is there a sequence $P_1, \ldots, P_n$ of procedures in Π such that all the instances in $outcomes_{P_1,\ldots,P_n}(I)$ satisfy Q?

First negative results: In the previous sections we have seen that most problems in our framework can be solved if we restrict ourselves to procedures with safe scope. Unfortunately, as the following result shows, this is not the case for the data-readiness problem.

Proposition 4. *The problem* DATA READINESS *is undecidable, even if Π is a set of procedures with safe scope.*

The proof is by reduction from the universal halting problem for Turing machines, along the lines of the proof used in [9] to show that termination of chase is undecidable. The proof uses Π to simulate each of the constraints being chased in the proof in [9].

Decidability for full tgds: In order to obtain decidability, we further restrict to sequences of procedures with safe scope and given by full tgds only, for which we can show the following result:

Proposition 5. *The problem of checking whether a boolean query Q holds in all outcomes of an instance I over a sequence $P_1, \ldots, P_n$ of procedures with safe scope and given by full tgds only, is decidable and in* EXPTIME.

This proposition suggests an algorithm for data readiness: one needs to guess a sequence of procedures, and then check whether the query is entailed in this sequence. Of course, we need a small-model property for the size of the sequence of procedures that we need to guess. Summing up, we have the following result.

Theorem 1. *For the cases where Π is a set of procedures with safe scope with output constraints comprising full tgds only,* DATA READINESS *is in* NEXPTIME.

6 Conclusion

In this paper we embark on the development of a framework that allows one to reason about database procedures based only on a high level description of what these procedures do. We have instantiated our framework in a relational setting, and only when procedures are guaranteed not to alter the schema of databases.

The proposed framework presents several opportunities for further research. One line of work would involve understanding how to represent outcomes of sequences of procedures, perhaps by means of knowledge bases or similar artifacts.

We also believe that our framework is general enough to allow reasoning on other data paradigms, or even across various different data paradigms. Our black-box abstraction could, for example, offer an effective way to reason about procedures involving unstructured text data, or even data transformations using machine-learning tools, as long as one can obtain some guarantees on the data outcomes of these tools.

References

1. D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data-aware process analysis: A database theory perspective. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 1–12, 2013.
2. M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *J. Comput. Syst. Sci.*, 28(1):29–59, 1984.
3. M. A. Casanova and V. M. P. Vidal. Towards a sound view integration methodology. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*, pages 36–47, 1983.
4. A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985.
5. R. Chirkova, J. Doyle, and J. L. Reutter. Assessing achievability of queries and constraints. *CoRR*, abs/1712.03438, 2017.
6. S. S. Cosmadakis and P. C. Kanellakis. Equational theories and database constraints. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 273–284, 1985.
7. S. S. Cosmadakis, P. C. Kanellakis, and M. Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. ACM*, 37(1):15–46, 1990.
8. D. Deutch and T. Milo. A quest for beauty and wealth (or, business processes for database researchers). In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–12. ACM, 2011.
9. A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 149–158. ACM, 2008.
10. P. G. Kolaitis, J. Panttaja, and W.-C. Tan. The complexity of data exchange. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 30–39, 2006.
11. J. C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56(3):154–173, 1983.
12. O. Savkovic, E. Marengo, and W. Nutt. Query stability in monotonic data-aware business processes. In *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, pages 16:1–16:18, 2016.
13. J. F. Sequeda. Ontology based data access: Where do the ontologies and mappings come from? In *AMW17.*, 2017.
14. V. Vianu. Automatic verification of database-driven systems: A new frontier. In *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, pages 1–13, 2009.