Complexity of Reachability for Data-aware Dynamic Systems

Parosh Aziz Abdulla*, C. Aiswarya†, Mohamed Faouzi Atig‡, Marco Montali§ and Othmane Rezine¶
* Uppsala University, Sweden parosh@it.uu.se
†Chennai Mathematical Institute, India aiswarya@cmi.ac.in
‡Uppsala University, Sweden mohamed_faouzi.atig@it.uu.se
§Free Univ. of Bozen/Bolzano, Italy montali@inf.unibz.it
¶Uppsala University, Sweden othmane.rezine@it.uu.se

***Abstract*—A formal model called database manipulating systems was introduced to model data-aware dynamic systems. Its semantics is given by an infinite labelled transition systems where a label can be an unbounded relational database. Reachability problem is undecidable over schemas consisting of either a binary relation or two unary relations. We study the reachability problem under schema restrictions and restrictions on the query language. We provide tight complexity bounds for different combinations of schema and query language, by reductions to/from standard formalism of infinite state systems such as Petri nets and counter systems. Our reductions throw light into the connections between these two seemingly unrelated models.**

I. Introduction

A lot of research in the verification community revolve around modeling of increasingly complex systems by classes of infinite state systems and studying their reachability. The verification of ubiquitous data-aware dynamic systems such as business processes is desirable along this line. A formal model called database manipulating systems (DMS) was introduced in [1] for their modelling and their model-checking was studied using under-approximation techniques. In this paper we study the precise complexity of the reachability problem, and how it is affected by varying the system parameters.

Database manipulating systems (DMS) have a dynamic relational database as an infinite memory. The actions of DMS will modify the database as per the rules of the actions: it first queries the database to retrieve some elements from it, and then updates the database by deleting some tuples involving the retrieved elements and/or adding new tuples to the database. The newly added tuples can contain elements that were not present in the database before. Thus they give rise to an infinite state system, where states are databases (or relational structures) over a relational schema. Though each database is finite, their data domain is unbounded, and the number of such databases can be infinite.

The infinite state system generated by our model can also be viewed laterally as generalizations of Kripke structures. While the state/world of a Kripke structure is labelled with a valuation of propositions, in our case it may be labelled with relations over an infinite domain (or databases). In other words, if the schema has only nullary relations or propositions, then a DMS will generate a Kripke structure. This view of relational Kripke structures has been proposed for modeling database-systems for verification purposes [28].

Reachability problem asks whether, given a DMS and an initial database, it is possible to reach another database from a target set of databases. The target set, for instance, can be the set of databases in which some proposition (nullary relation) is true. The propositional reachability problem is undecidable as one would expect. We showed in [1] that the undecidability holds as soon as the schema has at least two unary relations and the actions allow first-order queries, or, if the relational schema has a binary relation even when the action queries are conjunctive queries.

This poses the question of whether decidability can be achieved by weakening the schema and/or the query language used by actions. We answer this affirmatively in this paper and study the complexity of the reachability problem for various restrictions on schema and the query language.

Our techniques involve reductions to (variants of) Petri nets, throwing lights into the connections between these two seemingly different models of infinite state systems. As manifested in our reductions, the schema-restricted DMS serves as a succinct counter system.

Related work: The foundations of data management community has produced an impressive body of work on the verification of data-aware processes, that is, business processes synergically accounting for the control-flow and the data perspective [26], with the main goal of finding a suitable trade-off between expressiveness and decidability [9]. Several different formal models have been proposed towards this grand objective, tackling a plethora of concrete settings such as e-commerce applications [2], Web services exchanging data [17], the business artifact paradigm [15], [12], [5], business processes operating over relational databases with constraints [3], [1], [23] and over description logic knowledge bases [4], dynamic systems navigating over XML data [7], Situation Calculus [13], and multiagent systems [6], [24], [10].

Verification in these rich settings has been studied either by considering the system in its full generality but starting from a given initial state, or by considering the system independently from its initial state but limiting its ability of updating the data component. In this paper, we focus on processes operating over relational databases, and on the first kind of verification, which in general amounts to checking temporal/dynamic properties

over the relational transition system [28] induced by the database-driven dynamic system of interest starting from a given initial state.

Unsurprisingly, verification turns out to be undecidable even under severe restrictions, and decidability is typically regained by controlling how the process operates over the underlying data component [3], [23], [13], and/or by incorporating techniques from formal methods, such as sound abstraction [5] and under-approximation [1]. However, very little attention has been given to understanding how the schema of the data component, and the query language adopted by the process, relate to decidability of verification while considering the formal model of interest *in its full generality*.

In this paper, we focus on this problem and provide a full characterization of decidability and complexity of reachability under different restrictions on the schema of the DMS data component, and on the query language employed by the DMS process component. As for the schema, we consider in particular the presence of a single or multiple relations, and the arity of such relations. As for the query language, it ranges from conjunctive queries (and unions thereof) to full first-order queries with and without negation.

In the literature, few endeavors have been attempted in this direction. They typically refer to *unbounded* systems as a way to stress that no restriction is imposed on how the data component is updated by the process component, which can in fact even exploit the data component to accumulate an unbounded amount of information in each single state of the system. Specifically, [5] investigates unbounded artifact-centric systems, but only provides a partial model checking procedure, which does not provide any termination guarantee. In [23], so-called *case-centric* database-driven dynamic systems are studied, where each process execution (i.e., case) is associated to an explicit process instance identifier, in turn used to record all the data that relate to that case. Decidability of verification is then obtained for unboundedly many cases, but requiring two strong necessary conditions: first, that cases are isolated, i.e., do not co-refer the same tuples in the data component; second, that each case stores a bounded amount of information. This means that the system is still essentially bounded. Neither [5] nor [23] study how the obtained results are affected by the schema used to structure the data and the language used to query such data.

The closest approach to ours is [21], where the authors show that model checking unbounded artifact systems against a variant of first-order CTL is decidable when the artifact system under study is equipped with a single, unary relation. The result comes with a 3EXPTIME upper bound, but is not refined by focusing just on reachability properties, nor by studying the impact of the query language (e.g., by removing negation, essential to simulate zero-testing of a counter). Since the CARL formal model investigated in [21] is a fragment of the DMS model studied in this paper, our tight complexity results on reachability directly carry over to the setting of [21].

In [16], the authors study verification of a class of hierarchical artifact systems and use reduction to vector addition systems with states. Their setting is different since the initial database is not fixed, and in that sense is more parametric. Further they do not study schema restrictions / query language variations, but rather restrict the hierarchical nature for decidability.

II. PRELIMINARIES

Databases. We fix a (data) domain Δ, which is a countably infinite set of data values, acting as standard names. A relational schema $\mathcal{R}$ is a finite set $\{R_1/a_1, \ldots, R_n/a_n\}$ of relation names R_i, each coming with its own arity a_i. A database instance I over schema $\mathcal{R}$ and domain Δ is the union set $\cup_{i:1\leq i\leq n} R_i^I$, where $R_i^I \subseteq \{R_i\} \times \Delta^{a_i}$ represents the content of relation R_i in the database instance I. If I contains a tuple $\langle R_i, e_1, \ldots, e_{a_i}\rangle$, we write $R_i(e_1, \ldots, e_{a_i}) \in I$. A nullary relation $p/0$ (also known as proposition) can be either instantiated as the singleton set $\{p()\}$ or the empty set $\emptyset$. In the former case, we say the proposition is *true*, and write $p \in I$. In the latter case $p \notin I$ and we say p is *false*.

We denote the set of all database instances over $\mathcal{R}$ and Δ by $\mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \Delta)$. The *active domain* of I, denoted $\textsc{adom}(I)$, is the subset of Δ such that $e \in \textsc{adom}(I)$ iff e occurs in some tuple in I (i.e. there exist $\langle R_i, e_1, \ldots, e_{a_i}\rangle \in I$ such that $e = e_j$ for some $j : 1 \leq j \leq a_i$).

We may also view a database instance as a first-order structure where the vocabulary is specified by the schema. The active domain is the set of elements of this first-order structure which participates in some relation.

Given two database instances $I_1, I_2 \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \Delta)$, we define $I_1 + I_2$ to be the database instance $I \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \Delta)$ obtained by taking the relation-wise union. Similarly we define $I_1 - I_2$ where we take the relation-wise set difference. Simply put, for each relation R, we have $R^{I_1+I_2} = R^{I_1} \cup R^{I_2}$ and $R^{I_1-I_2} = R^{I_1} \setminus R^{I_2}$.

Queries. We use queries to access databases and extract data values of interest. Queries are expressed in *FOL with equality* over the schema $\mathcal{R}$ ($\mathtt{FOL}(\mathcal{R})$ for short). Let $\mathsf{Vars} = \{u, v, u_1, \ldots\}$ be the set of FO data-variables ranging over the data values in Δ. A $\mathtt{FOL}(\mathcal{R})$ *query* is given by the syntax:

$$\Phi ::= \mathsf{true} \mid R(u_1, \ldots, u_a) \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists u.\Phi \mid u_1 = u_2$$

where $R/a \in \mathcal{R}$, and u, u_i are variables from Vars. We use standard abbreviations like $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $\forall u.\Phi = \neg\exists u.\neg\Phi$, etc. We also denote with $\mathsf{Free\text{-}Vars}(\Phi)$ the set of free variables appearing in a query Φ.

A query is said to be an *existential FO query (*EXFO*)* if it is of the form $\exists u_1 \ldots \exists u_n\ \Phi$ where Φ is quantifier free. A query is said to be a *union of conjunctive queries (*UCQ*)* if it is of the form $\bigvee_{i\in\{1\ldots n\}} \bigwedge_{j\in\{1\ldots n_i\}} t_{i,j}$ where $t_{i,j}$ is a term of the form $R(u_1, \ldots, u_a)$ or $u_1 = u_2$ or true.

For a set $V \subseteq \mathsf{Vars}$, a *substitution* σ of V is a function that maps every variable in V to a value in Δ (i.e., $\sigma : V \to \Delta$). Given a substitution $\sigma : V \to \Delta$ and set $V' \subseteq V$, we define the restriction of σ on V' as the substitution $\sigma' : V' \to \Delta$ such that $\sigma'(u) = \sigma(u)$ for every $u \in V'$. We denote the restriction of σ to V' by $\sigma|_{V'}$.

Given a database instance I over $\mathcal{R}$ and Δ, a $\mathtt{FOL}(\mathcal{R})$ query Φ over $\mathcal{R}$, and a substitution $\sigma : \mathsf{Free\text{-}Vars}(\Phi) \to \textsc{adom}(I)$, we write $I, \sigma \models \Phi$ if the query Φ under the substitution σ

$$
\begin{aligned}
I, \sigma \models\ & \mathsf{true} \\
I, \sigma \models\ & R(u_1, \ldots, u_a) && \text{if } (e_1, \ldots, e_a) \in R^I \\
& \text{where } e_i = \sigma(u_i) \text{ for all } i : 1 \leq i \leq a. \\
I, \sigma \models\ & u_i = u_j && \text{if } \sigma(u_i) = \sigma(u_j). \\
I, \sigma \models\ & \neg Q && \text{if } I, \sigma \not\models Q \\
I, \sigma \models\ & Q_1 \wedge Q_2 && \text{if } I, \sigma \models Q_1, \text{ and } I, \sigma \models Q_2. \\
I, \sigma \models\ & \exists u.Q && \text{if there exists } e \in \textsc{adom}(I) \text{ such that} \\
& I, \sigma' \models\ Q \text{ where } \sigma'(u') = \begin{cases} e & \text{if } u' = u \\ \sigma(u') & \text{if } u' \neq u \end{cases}
\end{aligned}
$$

TABLE I: The semantics of $\mathtt{FOL}(\mathcal{R})$.

holds in database I. The semantics of $I, \sigma \models \Phi$ is standard. It is given in Table I. If Φ is a sentence, we may simply write $I \models \Phi$ instead of $I, \sigma \models \Phi$ since σ is empty.

Substitutions in database instances. Let $V \subseteq \mathsf{Vars}$ be a set of variables. Consider a substitution $\sigma : V \to \Delta$ that assigns each variable to an element from Δ. Let $I \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, V)$ be a database instance over schema $\mathcal{R}$ and the variables V. We define $\mathsf{Substitute}(I, \sigma) \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \Delta)$ to be the database instance obtained from I by substituting every occurrence of variable u by $\sigma(u)$, for each variable $u \in V$.

III. Data-base manipulating systems

A *Database-Manipulating System (DMS)* over domain Δ and schema $\mathcal{R}$ is a pair $\mathcal{S} = \langle I_0, \textsc{acts} \rangle$, where:

- $I_0 \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \Delta)$ is the *initial database instance*, with $\textsc{adom}(I_0) = \emptyset$. I_0 gives truth-values to nullary relations (aka propositions), and has empty non-nullary relations.
- $\textsc{acts}$ is a set of *(guarded) actions*. An action α is a tuple $\alpha = \langle \vec{u}, \vec{v}, \Phi, Del, Add \rangle$, where
  - $\vec{u}$ and $\vec{v}$ are disjoint finite subsets of Vars, respectively denoting *action parameters* and *input variables*.
  - Φ is a $\mathtt{FOL}(\mathcal{R})$ query, called the *guard* of α, such that $\vec{u} = \mathsf{Free\text{-}Vars}(\Phi)$.
  - $Del \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \vec{u})$ is a database instance over variables $\vec{u}$ and the schema $\mathcal{R}$.
  - $Add \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \vec{u} \uplus \vec{v})$ is a database instance over the variables $\vec{u} \uplus \vec{v}$ and $\mathcal{R}$, with $\vec{v} \subseteq \textsc{adom}(Add)$. The set $\vec{v}$ contains the so-called *input variables* of α.

Given an action $\alpha = \langle \vec{u}, \vec{v}, \Phi, Del, Add \rangle$, we sometimes simply write it as $(\Phi; Del; Add)$, colored to easily distinguish the Del and Add parts. Notice that $\vec{u}$ can be inferred from the guard ($\vec{u}$ is given by the free variables of Φ), and $\vec{v} = \textsc{adom}(Add) \setminus \vec{u}$.

Intuitively, a DMS operates as follows. It starts with the initial database instance I_0. At an instant, the DMS can update the current database instance by applying an action. An action is applied in three steps. In the first step the current database is queried using Φ to retrieve some elements of interest from its active domain. In the second step, some tuples involving the retrieved elements are removed from the current database, as dictated by the variable-database instance Del. Finally, new tuples may be added to the relations of the current database instance, as dictated by Add. The newly inserted tuples may contain fresh values that are not present in the active domain, and that are injected through the input variables. We give the formal execution semantics below.

Execution semantics. The execution semantics of a DMS $\mathcal{S} = \langle I_0, \textsc{acts} \rangle$ over $\mathcal{R}$ and Δ is defined in terms of a (possibly infinite) *configuration graph* $\mathcal{C}_\mathcal{S}$, which has the form of a relational transition system [28], [3]. Each configuration is a database instance $I \in \mathsf{DB\text{-}Inst\text{-}Set}(\mathcal{R}, \Delta)$.

Consider an action $\alpha = \langle \vec{u}, \vec{v}, \Phi, Del, Add \rangle$, and a substitution $\sigma : \vec{u} \uplus \vec{v} \to \Delta$. We say that σ is an *instantiating substitution* for α at a database instance I if it satisfies the following:

- for every variable $u_i \in \vec{u}$, $\sigma(u_i) \in \textsc{adom}(I)$ (action parameters are substituted with values from the current active domain);
- for every variable $v_i \in \vec{v}$, $\sigma(v_i) \notin \textsc{adom}(I)$ (input variables are substituted fresh values that are not present in the current active domain);
- $\sigma|_{\vec{v}}$ is injective (input variables are assigned to pairwise distinct values);
- $I, \sigma|_{\vec{u}} \models \Phi$ (the action guard is satisfied).

For a pair of database instances I, I', an action $\alpha = \langle \vec{u}, \vec{v}, \Phi, Del, Add \rangle \in \textsc{acts}$, and a substitution $\sigma : \vec{u} \uplus \vec{v} \to \Delta$, we have an edge $I \xrightarrow{\alpha:\sigma} I'$ in $\mathcal{C}_\mathcal{S}$, if

- σ is an instantiating substitution for α at I; and
- $I' = (I - \mathsf{Substitute}(Del, \sigma)) + \mathsf{Substitute}(Add, \sigma)$;

A *run* ρ of $\mathcal{S}$ is a sequence $I_0 \xrightarrow{\alpha_0:\sigma_0} I_1 \xrightarrow{\alpha_1:\sigma_1} I_2 \xrightarrow{\alpha_2:\sigma_2} I_3 \ldots I_n$ where I_0 is the initial database instance of $\mathcal{S}$. It is simply a finite path in the configuration graph starting at I_0.

Remark 1. The model of DMS that we introduce in this paper is slightly different than the one we had in [1]. In [1] we imposed that the input variables must be assigned values that have never been used in the history of a run. History-freshness was very crucial for the result of [1]. In the current model we relax this history-freshness requirement on the input variables, as the results of this paper will go through even without this assumption. We can achieve history-freshness in the new setting at the expense of two extra unary relations.

IV. Reachability problem and its variants

Since DMS are a very expressive model of computation, their verification is undecidable, even if we consider the simplest reachability problem. We state the problem below.

Problem:	FO reachability
Input:	A DMS $\mathcal{S}$, and FO sentence $\Phi_{\mathtt{tgt}}$
Question:	Does there exist a run $I_0 \xrightarrow{\alpha_0:\sigma_0} I_1 \xrightarrow{\alpha_1:\sigma_1} I_2 \xrightarrow{\alpha_2:\sigma_2} I_3 \ldots I_n$ such that $I_n \models \Phi_{\mathtt{tgt}}$

The ExFO reachability (resp. propositional reachability) problem is similar, we just require the input sentence $\Phi_{\mathtt{tgt}}$ to be an ExFO (resp. propositional) sentence.

Theorem 1. *[1] The reachability problem is undecidable.*

{Propositional, ExFO, FO } reachability		Schema of the form $\mathcal{R} = \{R_1/2, \dots\}$	$\mathcal{R} = \{R_1/1, R_2/1, \dots\}$
Action guards in	FO	Undecidable	Undecidable
	ExFO	Undecidable	Decidable
	UCQ	Undecidable	Decidable

TABLE II: Frontiers of decidability

FO Reachability		Schema of the form $\mathcal{R} = \{R_1/1, R_2/1, \dots\}$	$\mathcal{R} = \{R_1/1, P_1/0, \dots P_m/0\}$	$\mathcal{R} = \{P_1/0, \dots P_m/0\}$
Action guards in	FO	Undecidable	PSPACE-c	PSPACE-c
	ExFO	Cubic Ackermann, as hard as Petri net Reach.	PSPACE-c	PSPACE-c
	UCQ	Cubic Ackermann, as hard as Petri net Reach.	PSPACE-c	PSPACE-c

TABLE III: Complexity of FO reachability

The undecidability proved in [1] was for propositional reachability (i.e, when $\Phi_{\texttt{tgt}}$ is a single proposition). Further the DMS there assumed history-freshness. The undecidability proof there will carry over even without this assumption. Notice that the propositional reachability and ExFO reachability reduce to FO reachability trivially. We can also reduce the FO reachability to propositional reachability by adding a new proposition $p_{\texttt{tgt}}$ which will also act as the target sentence in the propositional reachability problem, and an action of the form $(\Phi_{\texttt{tgt}}; \emptyset; p_{\texttt{tgt}})$.

The undecidability holds as long as the schema contains a binary relation, even if the action guards are restricted to just UCQ. In fact, the undecidability holds even if the schema contains only unary relations if we allow the full power of action guards (FO). We study this in detail to understand the precise boundaries of (un)decidability – what restrictions on the schema/action guards can make the reachability problem decidable? We show that it is decidable in all other cases (cf. Table II). Furthermore, in the cases where we obtain decidability, we study the complexity of the reachability problem. Since we consider restricted languages for action guards, the FO reachability problem is not always reducible to propositional reachability problem. We provide the precise complexity characterization of FO reachability problem (Table III) and ExFO/ propositional reachability problem (Table IV) for the various combinations of schema restrictions and query languages[1].

The rest of the paper is devoted to proving the results in the above tables.

{Propositional, ExFO} reachability		Schema of the form $\mathcal{R} = \{R_1/1, R_2/1, \dots\}$	$\mathcal{R} = \{R_1/1, P_1/0, \dots P_m/0\}$	$\mathcal{R} = \{P_1/0, \dots P_m/0\}$
Action guards in	FO	Undecidable	PSPACE-c	PSPACE-c
	ExFO	2EXPSPACE-c	PSPACE-c	PSPACE-c
	UCQ	2EXPSPACE-c	PSPACE-c	PSPACE-c

TABLE IV: Complexity of propositional reachability and ExFO reachability

V. DMS OVER UNARY SCHEMA

In this section we will consider schema which consists of only unary and nullary relations. Let the schema be $\mathcal{R} = \{U_1/1, U_2/1, \dots, U_n/1, P_1/0, \dots P_m/0\}$. We denote the set of unary relations of $\mathcal{R}$ by $\mathcal{U}$ and the set of propositions of $\mathcal{R}$ by $\mathcal{P}$. That is $\mathcal{U} = \{U_1, U_2, \dots, U_n\}$ and $\mathcal{P} = \{P_1, \dots P_m\}$.

We will show that, given any DMS $\mathcal{S}$ with only ExFO action guards over $\mathcal{R}$, we can construct an exponential sized Petri net corresponding to $\mathcal{S}$ in Section V-A2. This Petri net will provide a faithful abstraction of the DMS $\mathcal{S}$. The FO reachability problem is then reduced to configuration reachability problem of this Petri net (Section V-A4), there by proving the decidability claimed in Table II. The ExFO reachability problem is reduced to coverability problem of this exponential sized Petri net, there by justifying the 2EXPSPACE upper bound claimed in Table IV (Section V-A3).

The FO reachability problem is shown to be at least as hard as the Petri net reachability in Section V-B1. We prove the lower bounds claimed in Table IV by a reduction from the reachability problem in chain counter systems in Section V-B2.

A. Upper bounds: from ExFO-DMS to Petri nets

1) Petri nets: We will first fix some notations that come handy in this section. Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, \dots\}$. Let S be a set. A multiset X of S, is a collection $X = \{\!\{x_1, x_2, \dots x_n\}\!\}$ where each $x_i \in S$, but they need not be pairwise distinct. It may also be equivalently seen as a mapping $X : S \to \mathbb{N}$ indicating the number of occurrences of each element of S in X. For $x \in X$, we sometimes write $X(x)$ as $|X|_x$. Given two multisets $X : S \to \mathbb{N}$ and $Y : S \to \mathbb{N}$, we denote their union by $X + Y$, i.e., $(X + Y)(x) = X(x) + Y(x)$ for all $x \in S$ and their intersection by $X \cap Y$, i.e., $(X \cap Y)(x) = \min\{X(x), Y(x)\}$ for all $x \in S$. The set of all multisets of S is denoted $\mathbb{N}^S$.

Recall that a Petri net is a tuple $\mathcal{PN} = \langle \mathsf{Places}, \mathsf{Trans}, \mathsf{in}, \mathsf{out} \rangle$ where Places is the set of places, Trans is the set of transitions, and $\mathsf{in}, \mathsf{out} : \mathsf{Trans} \to \mathbb{N}^{\mathsf{Places}}$ assign multisets of places to transitions describing the flow relation. A marking is map $\mathsf{m} : \mathsf{Places} \to \mathbb{N}$ indicating the number of tokens present in each place. We write $\mathsf{m} \xrightarrow{t} \mathsf{m}'$ for some transition $t \in \mathsf{Trans}$ if for all $\mathsf{p} \in \mathsf{Places}$ 1) $\mathsf{m}(\mathsf{p}) \geq |\mathsf{in}(t)|_{\mathsf{p}}$ where $|\mathsf{in}(t)|_p$ denotes the number of occurrences of p in $\mathsf{in}(t)$ and 2) $\mathsf{m}'(\mathsf{p}) = \mathsf{m}(\mathsf{p}) - |\mathsf{in}(t)|_{\mathsf{p}} + |\mathsf{out}(t)|_{\mathsf{p}}$.

The *coverability problem* of Petri net asks, given a Petri net $\mathcal{PN}$, an initial marking m_0 and a target marking m_f, does there exist a sequence of transitions $t_1 \dots t_n \in \mathsf{Trans}^*$ such that $\mathsf{m}_0 \xrightarrow{t_1} \mathsf{m}_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} \mathsf{m}'_f$ where $\mathsf{m}'_f(\mathsf{p}) \geq \mathsf{m}_f(\mathsf{p})$ for each $\mathsf{p} \in \mathsf{Places}$. The coverability problem is known to be EXPSPACE-COMPLETE [25], [20]. The *reachability problem*

[1]Cubic Ackermann complexity upperbound is established for Petri net reachability [19].

of Petri net asks, given a Petri net $\mathcal{PN}$, an initial marking m_0 and a target marking m_f, does there exist a sequence of transitions $t_1 \dots t_n \in \mathsf{Trans}^*$ such that $\mathsf{m}_0 \xrightarrow{t_1} \mathsf{m}_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} \mathsf{m}_f$. The reachability problem for Petri nets is decidable [22], [18], with a cubic Ackermann upper bound [19] .

2) Construction of the Petri net: We will first introduce some terminologies and definitions that we will be using in the construction. Consider a DMS over a schema consisting of n unary relations $\mathcal{U} = \{U_1, U_2, \dots, U_n\}$ and m nullary relations / propositions $\mathcal{P} = \{P_1, \dots P_m\}$. A *propositional state* is a subset of $\mathcal{P}$, indicating which propositions could be true at some database instance. A *unary state* is a non-empty subset of $\mathcal{U}$, indicating which unary relations an element in the active domain may be part of. Given a database instance I over $\mathcal{U}$ and $\mathcal{P}$, its propositional state is $\{P_i \mid I \models P_i\}$. Further, for an element $e \in \text{ADOM}(I)$, its unary state is $\{U_i \mid e \in U_i^I\}$. The set of possible unary states is $2^{\mathcal{U}} \setminus \emptyset$, and the set of propositional states is $2^{\mathcal{P}}$.

The construction of the Petri net is in two steps. First, we will obtain a new DMS $\mathcal{S}' = (I_0, \text{ACTS}')$ whose actions are written in a normal form called *type fully specified*. Next we will construct the Petri net corresponding to $\mathcal{S}'$.

In order to describe the transitions of the Petri net, it is convenient to assume that the guards of an action fully specify the unary type of every variable mentioned in the query. Let $\mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset$ be a unary state and let x be a free variable. We write $\mathsf{type}(x) = \mathsf{ustate}$ as a shorthand for $\bigwedge_{i:U_i \in \mathsf{ustate}} U_i(x) \wedge \bigwedge_{i:U_i \notin \mathsf{ustate}} \neg U_i(x)$. *Type fully-specified query* with free variables $x_1, \dots, x_k$ is an existential first-order query of the form

$$\mathsf{pstate} \wedge \exists x_{k+1} \dots \exists x_{k+\ell} \wedge_{i:1 \le i \le k+\ell} \mathsf{type}(x_i) = \mathsf{ustate}_i$$

where $\mathsf{pstate} \in 2^{\mathcal{P}}$ is a propositional state and $\mathsf{ustate}_i \in 2^{\mathcal{U}} \setminus \emptyset$ is a unary state for all $i : 1 \le i \le k + \ell$.

Given any DMS $\mathcal{S}$, we can convert it into a DMS $\mathcal{S}'$ with fully specified queries in exponential time with an exponential blow-up in the number of actions. Basically we convert the guard of an action into a disjunction of type fully-specified queries. For each disjunct, we write a separate action. The non-determinism of the DMS takes care of the disjunction. This does not change the schema. The DMS $\mathcal{S}$ and $\mathcal{S}'$ are equivalent, meaning that both of them have the same configuration graph. The DMS $\mathcal{S}'$ simply makes the action guards more explicit about the type.

Next we will construct the Petri net $\mathcal{PN}_{\mathcal{S}'} = \langle \mathsf{Places}, \mathsf{Trans}, \mathsf{in}, \mathsf{out} \rangle$ corresponding to the DMS $\mathcal{S}'$. Each configuration of the Petri net will represent an abstraction of a database instance. The Petri net transitions will bring in the effect of an action on these abstractions.

We will first describe the places of $\mathcal{PN}_{\mathcal{S}'}$. There will be two places corresponding to each proposition and also one place corresponding to each unary state. The place corresponding to a unary state $\mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset$ is denoted $\mathsf{plc}(\mathsf{ustate})$. The two places corresponding to a proposition $P \in \mathcal{P}$ are denoted $\mathsf{plc}(+P)$ and $\mathsf{plc}(-P)$. In any configuration, exactly one of $\mathsf{plc}(+P)$ and $\mathsf{plc}(-P)$ will contain a token indicating whether the proposition holds or not in a database instance. That is $\mathsf{Places} = \{\mathsf{plc}(+P) \mid P \in \mathcal{P}\} \cup \{\mathsf{plc}(-P) \mid P \in \mathcal{P}\} \cup \{\mathsf{plc}(\mathsf{ustate}) \mid \mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset\}$. Thus the number of places will be $2m + 2^n - 1$, where m is the number of propositions in the schema and n is the number of unary relations. Notice that the set of places only depend upon the schema.

Next we describe the transitions and the flow relations. The Petri net will have one transition t_α corresponding to each action α of $\mathcal{S}'$. That is, $\mathsf{Trans} = \{t_\alpha \mid \alpha \in \text{ACTS}'\}$. Recall that the DMS $\mathcal{S}'$ has only type-fully-specified queries. Consider an action $\alpha = \langle \vec{u}, \vec{v}, \Phi, Del, Add \rangle$ where $\vec{u} = u_1, \dots u_k$ and

$$\Phi = \mathsf{pstate} \wedge \exists u_{k+1} \dots \exists u_{k+\ell} \bigwedge_{i \in \{1,\dots,k+\ell\}} \mathsf{type}(u_i) = \mathsf{ustate}_i$$

is a type fully-specified ExFO query. We have a transition t_α in the Petri net corresponding to this action α. Its flow relations will reflect the effect of α. The inflow of t_α is the multiset given by $\mathsf{in}(t_\alpha) = \{\!\{\mathsf{plc}(+P) \mid P \in \mathsf{pstate}\}\!\} + \{\!\{\mathsf{plc}(-P) \mid P \notin \mathsf{pstate}\}\!\} + \{\!\{\mathsf{plc}(\mathsf{ustate}_i) \mid 1 \le i \le k + \ell\}\!\}$. For each $i : 1 \le i \le k$, let $\mathsf{ustate}'_i = \mathsf{ustate}_i \setminus \{U \mid U(u_i) \in Del\} \cup \{U \mid U(u_i) \in Add\}$. Further for each $v \in \vec{v}$ let $\mathsf{ustate}_v = \{U \mid U(v) \in Add\}$. Let $\mathsf{pstate}' = \mathsf{pstate} \setminus \{p \mid p \in Del\} \cup \{p \mid p \in Add\}$. The outflow of t_α is the multiset $\mathsf{out}(t_\alpha) = \{\!\{\mathsf{plc}(+P) \mid P \in \mathsf{pstate}'\}\!\} + \{\!\{\mathsf{plc}(-P) \mid P \notin \mathsf{pstate}'\}\!\} + \{\!\{\mathsf{plc}(\mathsf{ustate}'_i) \mid 1 \le i \le k \text{ and } \mathsf{ustate}'_i \neq \emptyset\}\!\} + \{\!\{\mathsf{plc}(\mathsf{ustate}_i) \mid k < i \le k + \ell\}\!\} + \{\!\{\mathsf{plc}(\mathsf{ustate}_v) \mid v \in \vec{v} \text{ and } \mathsf{ustate}(v) \neq \emptyset\}\!\}$.

This completes our construction of the Petri net. Next we argue that it provides a faithful abstraction of DMS $\mathcal{S}'$. A configuration/marking of the Petri net represents a database instance. In any configuration, there will be exactly one token corresponding to each proposition P, which could either be in the place $\mathsf{plc}(+P)$ indicating that the proposition is true in the database instance, or in the place $\mathsf{plc}(-P)$ otherwise. Further, the number of tokens in each unary state indicates the number of the elements in the database instance with the corresponding unary state.

The net effect of an action on the database instance include a) change of the unary state of some elements b) change of the propositional state c) removal of some elements from the active domain and d) addition of some elements with particular unary states into the active domain. Basically, for firing the transition t_α, we ensure that the guard is satisfied by checking there are enough tokens in the specified unary states. The existentially quantified bound variables are not changing their unary states. So transition just ensures that these elements exits by removing tokens from the respective unary states and putting them back. The propositional state is updated as dictated by the α. The free variables will be assigned to elements that may potentially change their unary state. The new unary state is described by ustate'_i. Notice that if an element is removed from the active domain, then ustate'_i will be $\emptyset$ and in this case the corresponding token is not put back by the transition. New elements may be inserted by the transition in the unary states given by ustate_v.

We make this connection formal by defining the type-counting abstraction of a database instance $\mathtt{tca}(I)$. It is defined as the multiset containing the propositional state of I, and the unary states of the elements in the active domain of I. That is, $\mathtt{tca}(I) = \{\!\{\mathsf{pstate}(I)\}\!\} + \{\!\{\mathsf{ustate}(e) \mid e \in \text{ADOM}(I)\}\!\}$.

We can think of $\mathtt{tca}(I)$ as a marking m in the Petri net. Whenever we have $I \xrightarrow{\alpha:\sigma} I'$ we have $\mathtt{tca}(I) \xrightarrow{t_\alpha} \mathtt{tca}(I')$ in the Petri net. Further, let the marking m be $\mathtt{tca}(I)$ for some database instance I. If $\mathsf{m} \xrightarrow{t_\alpha} \mathsf{m}'$ then in the DMS $\mathcal{S}'$ we have $I \xrightarrow{\alpha:\sigma} I'$ for some database instance I' with $\mathtt{tca}(I') = \mathsf{m}'$ and some substitution σ.

3) EXFO *reachability to coverability in Petri nets:* We will now describe the reduction from the EXFO reachability problem of DMS to coverability problem of Petri nets. Recall that the input to the EXFO reachability problem is a DMS $\mathcal{S} = (I_0, \text{ACTS})$ and an EXFO sentence $\Phi_{\mathtt{tgt}}$.

First of all, we will reduce the EXFO reachability to propositional reachability by adding a new proposition $p_{\mathtt{tgt}}$ to the schema which will also act as the target sentence in the propositional reachability problem, and an action of the form $(\Phi_{\mathtt{tgt}}; \emptyset; p_{\mathtt{tgt}})$. As we described before, we will construct the type fully-specified DMS $\mathcal{S}'$ from the modified input DMS and the corresponding Petri net $\mathcal{PN}_{\mathcal{S}'}$. To complete the reduction we need to specify the instance of the coverability problem which includes initial marking and the target marking in the Petri net. The initial marking $\mathsf{m}_0 = \{\!\{\mathsf{plc}(+P) \mid P \in \mathsf{pstate}(I_0)\}\!\} + \{\!\{\mathsf{plc}(-P) \mid P \notin \mathsf{pstate}(I_0)\}\!\}$. Note that the initial marking corresponds to $\mathtt{tca}(I_0)$. The target marking $\mathsf{m}_f = \{\!\{\mathsf{plc}(+\Phi_{\mathtt{tgt}})\}\!\}$ where $\Phi_{\mathtt{tgt}}$ is the target proposition given by the reachability problem. The proposition $p_{\mathtt{tgt}}$ is reachable from I_0 in $\mathcal{S}$ if and only if the proposition $p_{\mathtt{tgt}}$ is reachable from I_0 in $\mathcal{S}'$ if and only if the following coverability problem answers *yes*: given the Petri net $\mathcal{PN}_{\mathcal{S}'}$, the initial marking m_0 and the target marking m_f, does there exist a sequence of transitions $t_1 \dots t_n \in \mathsf{Trans}^*$ such that $\mathsf{m}_0 \xrightarrow{t_1} \mathsf{m}_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} \mathsf{m}'_f$ where $\mathsf{m}'_f(\mathsf{p}) \geq \mathsf{m}_f(\mathsf{p})$ for each $\mathsf{p} \in \mathsf{Places}$.

This concludes our reduction. We will now analyse the complexity of the reduction and the decision procedure. The construction of $\mathcal{S}'$ from $\mathcal{S}$ takes time exponential in the size of $\mathcal{S}$. Further the number of actions of $\mathcal{S}'$ can be exponential in the number of actions of $\mathcal{S}$ since we make them type fully-specified. The schema is the same for both. The number of places of the Petri net $\mathcal{PN}_{\mathcal{S}'}$ is exponential in the size of the schema. Further it has one transition of each action of $\mathcal{S}'$. Thus the size of the Petri net is exponential in the size of $\mathcal{S}$ and the construction takes only exponential time. Now the coverability problem of this exponential sized Petri net can be decided in double exponential space [25]. To summarize:

Theorem 2. EXFO *reachability in DMS with* EXFO *guards over schemas containing only unary and nullary relations is decidable in* 2EXPSPACE.

4) FO *reachability to reachability in Petri nets:* We now describe the reduction from the FO reachability problem of DMS to the reachability problem of Petri nets. Recall that the input to the FO reachability problem is a DMS $\mathcal{S} = (I_0, \text{ACTS})$ and a FO sentence $\Phi_{\mathtt{tgt}}$.

First, we will write the FO sentence $\Phi_{\mathtt{tgt}}$ as a finite disjunction of type fully-specified FO sentences. Each disjunct is a conjunction of terms stating conditions of the form

- a propositional state
- there are at least $\mathsf{lb}(\mathsf{ustate})$ elements of unary type ustate for $\mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset$
- there are at most $\mathsf{ub}(\mathsf{ustate})$ elements of unary type ustate for $\mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset$

Note that this is effectively possible since $\Phi_{\mathtt{tgt}}$ is a monadic FO sentence [8]. Further we will rewrite these disjuncts to more disjuncts where each disjunct is of the form

$$\mathsf{pstate} \wedge \bigwedge_{\mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset} \textit{count-of-}\mathsf{ustate} \bowtie n_{\mathsf{ustate}}$$

where $\bowtie \in \{=, \geq\}$. $\textit{count-of-}\mathsf{ustate} = n_{\mathsf{ustate}}$ is a formula stating that there are exactly n_{ustate} elements of unary type ustate, and $\textit{count-of-}\mathsf{ustate} \geq n_{\mathsf{ustate}}$ is a formula stating that there are at least n_{ustate} elements of unary type ustate.

For each of these disjunct we will synthesize an instance of the reachability problem. If at least one of these reachability problem answers yes, then we deduce a positive answer to the FO reachability problem of DMS, and a NO otherwise.

Consider a disjunct $\mathsf{pstate} \wedge \bigwedge_{\mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset} \textit{count-of-}\mathsf{ustate} \bowtie n_{\mathsf{ustate}}$ where $\bowtie \in \{=, \geq\}$. We will describe the reachability problem corresponding to this. As described before, we will construct the type fully-specified DMS $\mathcal{S}'$ and the corresponding Petri net $\mathcal{PN}_{\mathcal{S}'}$. We will then modify the Petri net $\mathcal{PN}_{\mathcal{S}'}$ such that a particular target marking is reachable if and only if the disjunct is satisfied. Towards this, we will add two new places called $\mathsf{phase1}$ and $\mathsf{phase2}$ to the Petri net. Initially $\mathsf{phase1}$ is marked and $\mathsf{phase2}$ is unmarked. At any point of time, the Petri net may non-deterministically choose to unmark $\mathsf{phase1}$ and mark $\mathsf{phase2}$. Once this is done, it cannot be undone. Towards this, we do the following.

- we add the place $\mathsf{phase1}$ to $\mathsf{in}(t_\alpha)$ as well as to $\mathsf{out}(t_\alpha)$ for each transition of $\mathcal{PN}_{\mathcal{S}'}$. Thus it forbids any transition of $\mathcal{PN}_{\mathcal{S}'}$ from firing once the activating token as moved to the place $\mathsf{phase2}$.
- we add a transition t_{switch} with $\mathsf{in}(t_{\mathsf{switch}}) = \{\!\{\mathsf{phase1}\}\!\}$ and $\mathsf{out}(t_{\mathsf{switch}}) = \{\!\{\mathsf{phase2}\}\!\}$. This transition enables the non-deterministic switching of phases.
- For each ustate such that $\textit{count-of-}\mathsf{ustate} \geq n_{\mathsf{ustate}}$ appears in the disjunct, we will have a transition t_{ustate} with $\mathsf{in}(t_{\mathsf{ustate}}) = \{\!\{\mathsf{plc}(\mathsf{ustate}), \mathsf{phase2}\}\!\}$ and $\mathsf{out}(t_{\mathsf{ustate}}) = \{\!\{\mathsf{phase2}\}\!\}$. This transition allows the Petri net to lose the extra tokens it might have accumulated in order to reach the exact marking we ask for in the reachability problem: exactly n_{ustate} tokens in the place $\mathsf{plc}(\mathsf{ustate})$.

The initial marking for the reachability problem is $\mathsf{m}_0 = \{\!\{\mathsf{plc}(+P) \mid P \in \mathsf{pstate}(I_0)\}\!\} + \{\!\{\mathsf{plc}(-P) \mid P \notin \mathsf{pstate}(I_0)\}\!\} + \{\!\{\mathsf{phase1}\}\!\}$. Note that the initial marking corresponds to $\mathtt{tca}(I_0)$ and it is in the first phase. The final marking for the reachability problem corresponding to the disjunct is $\mathsf{m}_f = \{\!\{\mathsf{phase2}\}\!\} + \{\!\{\mathsf{plc}(+P) \mid P \in \mathsf{pstate}(I_0)\}\!\} + \{\!\{\mathsf{plc}(-P) \mid P \notin \mathsf{pstate}\}\!\} + X$ where X is a multiset of $\{\mathsf{plc}(\mathsf{ustate}) \mid \mathsf{ustate} \in 2^{\mathcal{U}} \setminus \emptyset\}$ such that $X(\mathsf{plc}(\mathsf{ustate})) = n_{\mathsf{ustate}}$. That is, the final marking must satisfy the propositional state and must be in $\mathsf{phase2}$. Further, there must be exactly n_{ustate} many tokens in the place corresponding to the unary state ustate. Note that we work around the $\textit{count-of-}\mathsf{ustate} \geq n_{\mathsf{ustate}}$ constraint by allowing free decrement of tokens from these places in phase 2.

This completes our reduction. We summarise our result in the following theorem.

Theorem 3. FO *reachability in DMS with* ExFO *guards over schemas containing only unary and nullary relations is decidable in cubic Ackermann.*

B. Lower bounds for DMS over unary schema

For establishing lower bounds we give reductions from hard problems to reachability in DMS. The DMS that we construct in the reduction uses only UCQ guards in the actions.

1) FO *reachability is as hard as Petri net reachability:* Given an instance of a Petri net reachability problem, say $\mathcal{PN} = \langle \mathsf{Places}, \mathsf{Trans}, \mathsf{in}, \mathsf{out}\rangle$, initial marking m_0 and target marking m_f, we construct, in linear time, an instance of FO reachability problem in DMS with UCQ guards as follows. The schema consists of one proposition P_{init} and $|\mathsf{Places}|$ many unary relations $\{U_{\mathsf{p}} \mid \mathsf{p} \in \mathsf{Places}\}$. The initial database instance I_0 has the proposition P_{init} set to true and an empty active domain. A special action α_0 populates the database instance to match the initial marking: $\alpha_0 = (P_{\mathsf{init}}; P_{\mathsf{init}}; \{U_{\mathsf{p}}(v^i_{\mathsf{p}}) \mid \mathsf{p} \in \mathsf{Places} \text{ and } 1 \leq i \leq |\mathsf{m}_0|_{\mathsf{p}}\})$. Further it has an action $\alpha_t = (\Phi_t, Del_t, Add_t)$ corresponding to each transition $t \in \mathsf{Trans}$ in $\mathcal{PN}$, where $\Phi_t = \neg P_{\mathsf{init}} \wedge \bigwedge_{\mathsf{p}\in\mathsf{Places}} \bigwedge_{1\leq i\leq|\mathsf{in}(t)|_{\mathsf{p}}} U_{\mathsf{p}}(u^i_{\mathsf{p}})$, $Del_t = \{U_{\mathsf{p}}(u^i_{\mathsf{p}}) \mid \mathsf{p} \in \mathsf{Places}, 1 \leq i \leq |\mathsf{in}(t)|_{\mathsf{p}}\}$, and $Add_t = \{U_{\mathsf{p}}(v^i_{\mathsf{p}}) \mid \mathsf{p} \in \mathsf{Places}, 1 \leq i \leq |\mathsf{out}(t)|_{\mathsf{p}}\}$. Finally the FO sentence $\Phi_{\mathtt{tgt}}$ is given by $\neg P_{\mathsf{init}} \wedge (\exists u^i_{\mathsf{p}})_{\mathsf{p}\in\mathsf{Places},1\leq i\leq|m_f|_{\mathsf{p}}}$ (

$$\bigwedge_{\mathsf{p}\in\mathsf{Places}} \bigwedge_{1\leq i\leq|m_f|_{\mathsf{p}}} U_{\mathsf{p}}(u^i_{\mathsf{p}}) \wedge \forall u \bigvee_{\mathsf{p}\in\mathsf{Places}} \bigvee_{1\leq i\leq|m_f|_{\mathsf{p}}} u = u^i_{\mathsf{p}} \ ).$$

Our reduction is complete.

2) Propositional reachability is 2-ExpSpace hard: We give a reduction from the control state reachability problem in chain counter systems introduced in [14] with exponentially many counters. We describe the latter below, in the special case that is needed for our reduction.

A **chain counter system with exponentially many counters** is described by a tuple $\mathcal{CS} = \langle n, \mathsf{States}, \mathsf{Init}, \mathsf{Acc}, \mathsf{Op}, \mathsf{Trans}\rangle$, where the system has 2^n many counters, States is the set of control states, Init is the initial control state, Acc is the set of final control states, and $\mathsf{Trans} \subseteq \mathsf{States} \times \mathsf{Op} \times \mathsf{States}$ is the set of transitions where the set of operation is given by $\mathsf{Op} = \{\mathsf{inc}, \mathsf{dec}, \mathsf{next}, \mathsf{prev}, \mathsf{first?}, \mathsf{notFirst?}, \mathsf{last?}, \mathsf{notLast?}\}$.

The counters are linearly ordered. At any configuration, exactly one counter can be acted upon, call it curr-counter. On taking a transition at a configuration, the operation is applied to curr-counter if it is inc or dec. A transition with operation first? (resp. last?) can be taken only if curr-counter is the first (resp. last). Similarly a transition with operation notFirst? (resp. notLast?) can be taken only if curr-counter is not the first (resp. not the last). The operation next (resp. prev) is used to set the next counter (resp. previous counter) as curr-counter.

The semantics is as expected. The control state reachability problem of chain counter systems asks, given a chain counter system $\mathcal{CS}$ and state s of $\mathcal{CS}$, whether s can be reached by a run of the $\mathcal{CS}$.

Theorem 4. *[14] The control state reachability problem of chain counter system with exponentially many counters is* 2-ExpSpace-*Complete.*

Reduction. Given a chain counter system $\mathcal{CS} = \langle n, \mathsf{States}, \mathsf{Init}, \mathsf{Acc}, \mathsf{Op}, \mathsf{Trans}\rangle$, we will construct a DMS $\mathcal{S}_{\mathcal{CS}}$ of polynomial size over a schema with only unary relations and propositions. First we describe the unary relations. There are $2n$ unary relations $Q_1, \ldots, Q_n$ and $\bar{Q}_1, \ldots, \bar{Q}_n$. The relations Q_i and $\bar{Q}_i$ are complementary in the sense that, an element in the active domain belongs to Q_i if and only if it does not belong to $\bar{Q}_i$. Thus there are exponentially many possible types for an element. Indeed, the type of an element can be seen as a binary number, where the ith bit is 1 if and only if it belongs to Q_i (and 0 if and only if it belongs to $\bar{Q}_i$). We will need one more unary relation mkd that we use in the simulation of transitions. The use of it will be clear when we describe the simulations.

Each counter in the chain system can be represented in binary in n bits. The value of a counter c is determined by the number of elements i whose type corresponds to the binary representation of c.

Further the schema of DMS $\mathcal{S}_{\mathcal{CS}}$ will have $\mathsf{Poly}(n, |\mathsf{States}|, |\mathsf{Trans}|)$ many propositions. The values of the propositions encode the current state and the current counter. To encode the current counter we use n propositions $P_1, \ldots P_n$. The truth assignment of these n propositions can be seen as a binary number indicating the index of the current counter that is being acted upon. Further there are propositions of the form P_s for $s \in \mathsf{States}$ which is true if and only if the current state of the $\mathcal{CS}$ is s. Moreover, we will have $(n-1) \times |\mathsf{Trans}|$ propositions of the form P^i_τ for $\tau \in \mathsf{Trans}$ and $i \in \{2, \ldots, n\}$. These propositions acts as intermediate states during the simulation of the transition τ. Among all the propositions of the form P_s and P^i_τ, one and only one can be true at any time. Thus the schema $\mathcal{R} = \{Q_1/1, \ldots, Q_n/1, \bar{Q}_1/1, \ldots, \bar{Q}_n/1, \mathsf{mkd}/1, P_1/0, \ldots P_n/0\} \cup \{P_s/0 \mid s \in \mathsf{States}\} \cup \{P^i_\tau \mid \tau \in \mathsf{Trans}, 2 \leq i \leq n\}$.

Next we describe how we simulate the transitions. For each transition we give below a gadget representing the simulation. The vertices of these gadgets are labelled by propositions meaning that in order to take any outgoing edge from such a vertex, the corresponding proposition must be true. Having said this, we do not explicitly write it in the transition guard for the sake of conciseness. The edge labels represent the actions of the DMS $\mathcal{S}_{\mathcal{CS}}$. A special unary relation mkd is used by the gadgets to mark a particular element through the various steps in the gadget.

The transition $\tau = (s, \mathsf{inc}, t)$ corresponds to the gadget shown in Fig. 1. Basically it adds a new element to the database instance whose type corresponds to the current counter, which is given by the propositional state of the $P_1 \ldots P_n$. The new element is added to the unary relation Q_i if P_i is true, and to $\bar{Q}_i$ if P_i is false. Further, the proposition P_s must be true in order to enter the gadget, and when it exits the gadget it is set to false and P_t is set to true.

The gadget corresponding to the transition $\tau = (s, \mathsf{dec}, t)$ is given Fig. 2. It is similar in structure to that of inc, but it

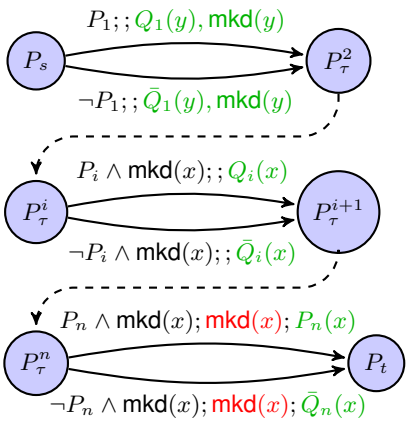

Fig. 1: Gadget simulating $\tau = (s, \mathsf{inc}, t)$

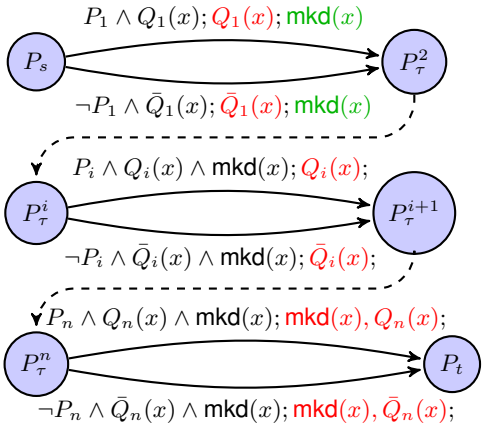

Fig. 2: Gadget simulating $\tau = (s, \mathsf{dec}, t)$

removes an element from the active domain. Further the type of the removed element corresponds to the current counter, which is given by the propositional state of the $P_1 \dots P_n$. This is ensured by the conjunction of all the action guards in the taken in a path from P_s to P_t. Notice that, by using the special unary relation mkd we ensure that the same element is affected in each action.

Next we give the gadgets for transitions of the form $\tau = (s, \mathsf{first?}, t)$ and $\tau = (s, \mathsf{notFirst?}, t)$ respectively in Fig.3 and Fig.4. Basically the former checks that the propositional state of $P_1 \dots P_n$ corresponds to binary 0 representing the first counter, and the latter checks that at least one bit is 1 (P_i is true). The gadgets for transitions of the form $\tau = (s, \mathsf{last?}, t)$ and $\tau = (s, \mathsf{notLast?}, t)$ are analogous.

The gadgets for next and prev are more tricky. A next action corresponds to incrementing the binary number indicated by the propositional state of $P_1 \dots P_n$. Let i be the first bit to be flipped by the increment operation (assuming the nth bit is the least significant bit). That is, P_i was set to false before, and will be set to true now, and each P_{i+j} was set to true before, and will be set to false now. For each value of i we have a unique path from the source state s to the target state t as given in the gadget for $\tau = (s, \mathsf{next}, t)$ in Fig. 5. Notice that the path taking the ith outgoing edge makes sure that P_i was set to false before and all P_{i+j} was set to true before.

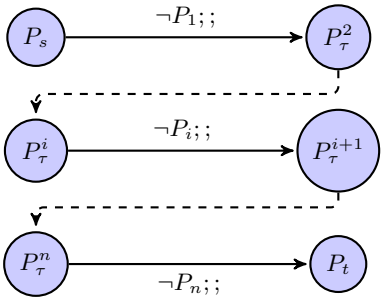

Fig. 3: Gadget for $\tau = (s, \mathsf{first?}, t)$

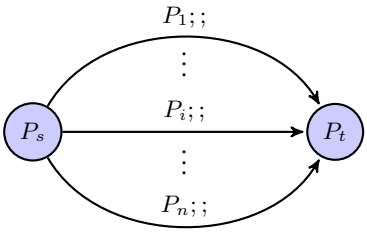

Fig. 4: Gadget for $\tau = (s, \mathsf{notFirst?}, t)$

Further when P_t is reached, P_i is set to true and all P_{i+j} is set to false. The gadget for $\tau = (s, \mathsf{prev}, t)$ is analogous, given in Fig. 6.

We have now described the schema and actions of the DMS $\mathcal{S}_{\mathcal{CS}}$. Notice that $\mathcal{S}_{\mathcal{CS}}$ employs only UCQ guards in actions. It remains to describe the I_0 as well as the target proposition to complete the reduction. The initial database will have all the propositions set to false but P_{Init}, the proposition indicating the initial state of the chain counter system. If the control state reachability problem gives $s \in \mathsf{States}$ as the target state of the

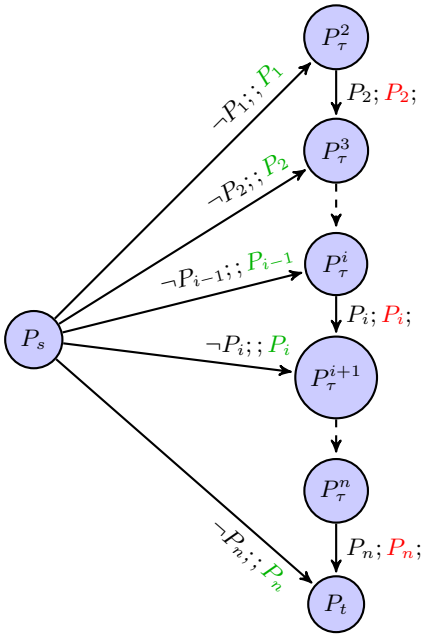

Fig. 5: Gadget for $\tau = (s, \mathsf{next}, t)$

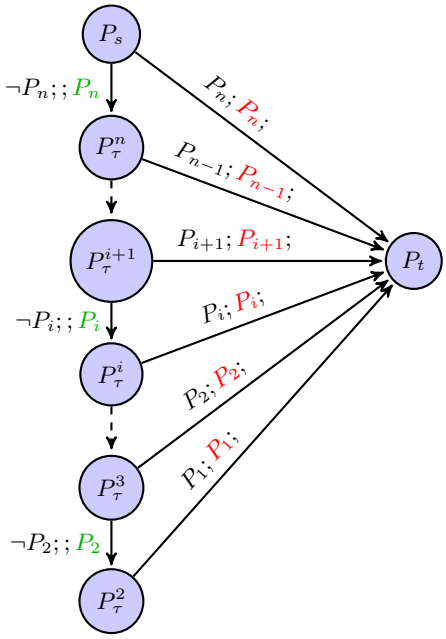

Fig. 6: Gadget for $\tau = (s, \mathsf{prev}, t)$

counter system, we will set the proposition P_s as the target proposition $\Phi_{\mathtt{tgt}}$ for the propositional reachability problem in $\mathcal{S}_{\mathcal{CS}}$. This completes our polynomial time reduction. To summarize:

Theorem 5. *Propositional reachability problem of DMS with* UCQ *action guards over schemas with only unary relations and propositions is* 2EXPSPACE*-hard.*

This proves all the 2EXPSPACE-hardness stated in Table IV. Along with the upper bound stated in Theorem 2, we conclude all the 2EXPSPACE-completeness claimed in Table IV.

VI. DMS OVER AT MOST ONE UNARY RELATION

For DMS over schemas with at most one unary relation (i.e, schemas of the form $\mathcal{R} = \{R_1/1, P_1/0, \ldots P_m/0\}$ or $\mathcal{R} = \{P_1/0, \ldots P_m/0\}$), the $\{\mathsf{FO}, \mathsf{ExFO}, \text{propositional }\}$-reachability problem are PSPACE complete, no matter which language is chosen for action guards.

We will show in Section VI-A the PSPACE upper bound for FO reachability problem of DMS with FO action guards over a schema of the form $\mathcal{R} = \{R_1/1, P_1/0, \ldots P_m/0\}$, thereby implying all the other PSPACE upper bounds claimed in Tables III and IV. In Section VI-B we show the PSPACE-hardness for propositional reachability of DMS with UCQ action guards over a schema of the form $\mathcal{R} = \{P_1/0, \ldots P_m/0\}$, thereby validating all PSPACE lower bounds claimed in Tables III and IV.

A. PSPACE *upper bound*

We first show that the problem is in PSPACE even for DMS with full FO guards.

We will prove the upper bound by a reduction to the control state reachability problem in a one-counter automaton. The latter is known to be in NL, since if it is reachable, it is reachable by a short path of polynomial length [11], [27].

Given a DMS over schema $\mathcal{R} = \{R_1/1, P_1/0, \ldots P_m/0\}$, our idea is to construct a one-counter automaton of exponential size. Instead of constructing it entirely, we will perform the non-deterministic reachability algorithm on-the-fly, showing that the problem can be solved in PSPACE.

In our reduction, the propositional state of the DMS will correspond to the control locations of the one counter automaton. Further the size of the active domain will correspond to the counter value. Since there is only a single unary relation, the counter value faithfully represents the number of elements participating in the unary relation. The initial control location is exactly the propositional state of I_0, and the initial counter value is 0.

Corresponding to each action, we will have a transition. Note that every FO query corresponds to a cardinality constraint on the relation. An FO query Φ can ask for the existence of (exactly, or at least, or at most) k elements in the unary relation, where k is polynomially bounded by the length of the query Φ. Thus for each action, there is a gadget which first checks that the guard is satisfied by making sure that the current state along with the counter value will satisfy the guard. For instance if the guard required the existence of exactly k elements, then the counter will be decremented k times, followed by a zero-test, followed by k increments. Notice that k is bounded by the length of the query Φ. To continue with the action, then some m elements may be deleted and some n elements may be added to the relation. The gadget implements this by decrementing m times and then incrementing n times. The target state will correspond to the new propositional state after updating the truth values of the propositions as per the action.

The on-the-fly reachability algorithm will check whether any propositional state where the proposition $p_{\mathtt{tgt}}$ is true is reachable. This can be done non-deterministically in polynomial space, and hence by Savitch's theorem, is in PSPACE.

Theorem 6. FO *reachability problem of DMS with* FO *action guards over schema of the form* $\mathcal{R} = \{R_1/1, P_1/0, \ldots P_m/0\}$ *is in* PSPACE*.*

B. PSPACE*-hardness*

Next we show that the complexity is optimal by giving a matching lower bound. The PSPACE hardness holds already for the propositional reachability problem of DMS with UCQ action guards over a schema with only nullary relations. We show PSPACE hardness by a reduction from the intersection non-emptiness problem of m finite state automata.

Suppose the input to the intersection non-emptiness problem is m finite state automata $\mathcal{A}_1, \ldots \mathcal{A}_m$, over a finite alphabet Σ. Let $\mathcal{A}_i = \langle \mathsf{States}_i, \mathsf{Trans}_i, \mathsf{init}_i, \mathsf{Acc}_i \rangle$. Without loss of generality, we assume that the set of states States_i are disjoint. We also assume that $|\mathsf{Acc}_i| = 1$, by letting the automata to be non-deterministic. The question is whether there exists a word $w \in \Sigma^*$ such that w is accepted by all $\mathcal{A}_i$.

We construct a DMS $\mathcal{S}$ with UCQ action guards as follows. The schema $\mathcal{R}$ consists of only nullary relations (or proposi-

tions). For each state $s \in \bigcup_i \mathsf{States}_i$, we have a proposition P_s in $\mathcal{R}$. Further, for each letter $a \in \Sigma$, we have P_a in $\mathcal{R}$. Further $\mathcal{R}$ also has $P_{\mathcal{A}_i}$ for each $i : 1 \leq i \leq m$ and two special values $P_\$$ and P_{Acc}. That is, $\mathcal{R} = \{P_s/0 \mid s \in \bigcup_i \mathsf{States}_i\} \cup \{P_a/0 \mid a \in \Sigma\} \cup \{P_{\mathcal{A}_i}/0 \mid 1 \leq i \leq m\} \cup \{P_\$/0, P_{\mathsf{Acc}}/0\}$.

Initially, the propositions P_{init_i} and the special proposition $P_\$$ are set to true, and all other propositions are set to false. This defines I_0. This means that all automata are in their respective initial states, and that it is ready to start a new round. The actions ACTS of DMS, are defined as follows. For each $a \in \Sigma$, it has actions of the form

1) $(P_\$; P_\$; P_a, P_{\mathcal{A}_1})$. It guesses the next letter of the word accepted commonly by all automata, and gives the control to automaton $\mathcal{A}_1$.
2) $(P_{\mathcal{A}_i} \wedge P_s \wedge P_a; P_{\mathcal{A}_i}, P_s; P_{s'}, P_{\mathcal{A}_{i+1}})$, where $1 \leq i < m$, if $(s, a, s') \in \mathsf{Trans}_i$. It simulates a transition of $\mathcal{A}_i$ on the letter a and gives the control to $\mathcal{A}_i$
3) $(P_{\mathcal{A}_m} \wedge P_s \wedge P_a; P_{\mathcal{A}_m}, P_s, P_a; P_{s'}, P_\$)$ if $(s, a, s') \in \mathsf{Trans}_m$. It simulates a transition of $\mathcal{A}_m$ on the letter a and sets $P_\$$ indicating that a new round may start.

Further, we add to the set of actions the action $(P_\$ \wedge \bigwedge_{1 \leq i \leq m} p_{s_i}; ; P_{\mathsf{Acc}})$ if $(s_i \in \mathsf{Acc}_i)_{1 \leq i \leq m}$. Notice that since $|\mathsf{Acc}_i| = 1$ for all i, we add only one action at this step. The total number of actions is $1 + |\Sigma| + \sum_i |\mathsf{Trans}_i|$. Finally, the target proposition for the reachability problem is simply P_{Acc}.

By virtue of our construction, at any point of time, there is exactly one state per automaton that is set to true. Any run of $\mathcal{S}$ will simulate a synchronous step of the automata $\mathcal{A}_i$ in a sequential fashion: It first guesses the next letter composing the commonly accepted word, and simulates a transition of each automaton on that letter one after the other. The proposition $P_\$$ indicates that none of the automata are active currently, meaning that it is ready to start a new round of sequential simulation. The control state description P_{Acc} is true only if $P_\$$ is true and all automata end up in accepting states. Thus the proposition P_{Acc} can be satisfied if, and only if, the intersection of the automata $\mathcal{A}_i$ is non-empty. Note that, the size of $\mathcal{S}$ is polynomial in the input and the reduction is also polynomial time.

We summarize the results below:

Theorem 7. *Propositional reachability of DMS with* UCQ *action guards over a schema of the form* $\mathcal{R} = \{P_1/0, \ldots P_m/0\}$ *is* PSPACE*-hard.*

This, along with Theorem 6 justifies all the PSPACE-completeness claimed in Tables III and IV.

VII. Conclusions

We have studied the boundaries of decidability of the reachability problem in DMS — a formal model for evolving database systems, along possible restrictions of schema and the query language used for updates. Further we have given tight complexity bounds for the reachability problem in the decidable cases, as summarized in Tables II, III and IV.

As a possible direction for future research, it would be interesting to study the decidability status and complexity of more involved linear/branching-time temporal properties while subject to similar restrictions on schema and query language.

References

[1] Parosh Aziz Abdulla, C. Aiswarya, Mohamed Faouzi Atig, Marco Montali, and Othmane Rezine. Recency-bounded verification of dynamic database-driven. In *PODS*, pages 195–210. ACM, 2016.

[2] Serge Abiteboul, Victor Vianu, Bradley Fordham, and Yelena Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000.

[3] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *PODS*, 2013.

[4] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic Knowledge and Action Bases. *JAIR*, 46, 2013.

[5] Francesco Belardinelli. Verification of non-uniform and unbounded artifact-centric systems: decidability through abstraction. In *AAMAS*, 2014.

[6] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. An abstraction technique for the verification of artifact-centric systems. In *KR*, 2012.

[7] Mikolaj Bojanczyk, Luc Segoufin, and Szymon Torunczyk. Verification of database-driven systems via amalgamation. In *PODS*, 2013.

[8] Egon Brger, Erich Grdel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

[9] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data-aware process analysis: A database theory perspective. In *PODS*. ACM Press, 2013.

[10] Diego Calvanese, Giorgio Delzanno, and Marco Montali. Verification of relational multiagent systems with data types. In *AAAI*. AAAIP, 2015.

[11] Dmitry Chistikov, Wojciech Czerwiński, Piotr Hofman, Michał Pilipczuk, and Michael Wehar. Shortest paths in one-counter systems. In *Proceedings of FOSSACS 2016*, pages 462–478. Springer, 2016.

[12] E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. In *ICDT*, 2011.

[13] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *AIJ*, 2016.

[14] Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. *Logical Methods in Computer Science*, 12(3), 2016.

[15] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[16] Alin Deutsch, Yuliang Li, and Victor Vianu. Verification of hierarchical artifact systems. In *PODS*, pages 179–194. ACM, 2016.

[17] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

[18] S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 267–281. ACM, 1982.

[19] Jerome Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *LICS '15*, pages 56–67, 2015.

[20] R. Lipton. The reachability problem requires exponential time. Technical report, Yale University, 1976. Technical Report 62.

[21] Alessio Lomuscio and Jakub Michaliszyn. Model checking unbounded artifact-centric systems. In *KR*. AAAIP, 2014.

[22] Ernst W. Mayr. An algorithm for the general petri net reachability problem. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 238–246. ACM, 1981.

[23] Marco Montali and Diego Calvanese. Soundness of data-aware, case-centric processes. *Int. Journal on Software Tools for Technology Transfer*, 2016.

[24] Marco Montali, Diego Calvanese, and Giuseppe De Giacomo. Verification of data-aware commitment-based multiagent systems. In *AAMAS*, 2014.

[25] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223 – 231, 1978.

[26] Manfred Reichert. Process and data: Two sides of the same coin? In *OTM*, volume 7565 of *LNCS*, pages 2–19. Springer, 2012.

[27] Leslie G. Valiant and Michael S. Paterson. Deterministic one-counter automata. *Journal of Computer and System Sciences*, 10(3):340 – 350, 1975.

[28] Moshe Y. Vardi. Model checking for database theoreticians. In *ICDT*, volume 3363 of *LNCS*, pages 1–16. Springer, 2005.