On the Complexity of Evaluating Regular Path Queries over Linear Existential Rules

Meghyn Bienvenu[1] and Michaël Thomazo[2]

[1] CNRS, Université de Montpellier, & Inria, Montpellier, France
meghyn@lirmm.fr
[2] Inria, France
michael.thomazo@inria.fr

Abstract. In the setting of ontology-mediated query answering, a query is evaluated over a knowledge base consisting of a database instance and an ontology. While most work in the area focuses on conjunctive queries, navigational queries are gaining increasing attention. In this paper, we investigate the complexity of evaluating the standard form of navigational queries, namely two-way regular path queries, over knowledge bases whose ontology is expressed by means of linear existential rules. More specifically, we show how to extend an approach developed for DL-Lite$_\mathcal{R}$ to obtain an exponential-time decision procedure for linear rules. We prove that this algorithm achieves optimal worst-case complexity by establishing a matching ExpTime lower bound.

1 Introduction

Ontology-mediated query answering (OMQA) has generated a lot of interest in the last years as a promising way of facilitating access to data (see [4] for a recent survey). In the OMQA approach, the ontology serves to define a conceptual view of an application domain, introducing a convenient vocabulary for query formulation and providing background knowledge that is exploited at query time to obtain the complete set of answers. So far, the vast majority of research on OMQA has considered user queries in the form of conjunctive queries (CQs), which are a standard query language for relational databases. However, in numerous application scenarios, data can naturally be seen as graphs, in which case so-called *navigational queries* are considered more suitable. The basic navigational query language is regular path queries (RPQs) [11], which allow one to find paths whose labels conform to a given regular language.

In recent years, the problem of answering navigational queries in the setting of OMQA has begun to be explored, first for ontologies formulated in highly expressive description logics (DLs) of the $\mathcal{Z}$ family [8,9,10], then for rich Horn DLs like Horn-$\mathcal{SROIQ}$ [18], and more recently, for lightweight DLs like DL-Lite$_\mathcal{R}$ and $\mathcal{EL}$ [19,5]. The latter DLs, which underlie the OWL 2 QL and EL profiles, are the most relevant for OMQA due to their favourable computational properties. In addition to plain RPQs, this line of work has also considered richer navigational languages like conjunctive RPQs (which extend both RPQs and CQs) and extensions with nesting and/or negation [3,6,15]. Although much work remains to be done in developing and implementing efficient algorithms, the complexity landscape for answering various forms of path queries over DL knowledge bases is now rather well understood. The same cannot be said for ontologies formulated by means of decidable classes of existential rules (like linear and guarded rulesets), which constitute another important class of ontology languages [7,1]. A key feature that distinguishes existential rules from DLs is the possibility of using predicates of arity greater than two. Since regular path queries are defined only with respect to unary and binary predicates, one might wonder whether they make sense in higher arity settings. We argue however that unary and binary predicates form the backbone of real-world ontologies (irrespective of the choice of ontology language), and it is desirable to be able to use some higher-arity predicates without losing any expressivity in the query language.

In this paper, we take a step towards a better understanding of the combination of navigational query languages and existential rules by studying the complexity of answering two-way RPQs in the presence of linear rules, a well-studied class of existential rules that are a natural generalization of the DL-Lite description logics. After introducing the necessary background, we show how to adapt the RPQ algorithm for DL-Lite proposed in [5] to the setting of linear rules. Unfortunately, our adaptation incurs an exponential blow-up with respect to the maximum predicate arity. We can nevertheless show that the obtained algorithm is worst-case optimal, as RPQ answering is EXPTIME-complete in combined complexity.

2 Preliminaries

We adopt the notation of [13]. The notions of constants, function symbols and predicate symbols are standard. Each function or predicate symbol is associated with a nonnegative integer arity. Variables, terms, substitutions, atoms, first-order formulae, sentences, interpretations (*i.e.*, structures), and models are defined as usual. By a slight abuse of notation, we often identify a conjunction with the set of its conjuncts. Furthermore, we often abbreviate a vector of terms $t_1, \ldots, t_n$ as $\mathbf{t}$, and define $|\mathbf{t}| = n$. By $\varphi\sigma$ we denote the result of applying a substitution σ to φ. A term, atom, or formula is *ground* if it does not contain variables; a *fact* is a ground atom. A term t' is a subterm of a term t if $t' = t$ or $t = f(\mathbf{s})$ where f is a function and t' is a subterm of some $s_i \in \mathbf{s}$. A term s is *contained* in an atom $p(\mathbf{t})$ is $s \in \mathbf{t}$, and s *occurs* in $p(\mathbf{t})$ if s is a subterm of some term $t_i \in \mathbf{t}$; thus, if s is contained in $p(\mathbf{t})$, s occurs in $p(\mathbf{t})$, but the converse may not hold. A term s is *contained* (resp. *occurs*) in a set of atoms I if s is contained (resp. occurs) in some atom in I. Let $T = \{t_1, \ldots, t_n\}$ be a set of terms. A term t is *generated by* T if *(i)* $t \in T$ or *(ii)* $t = f(x_1, \ldots, x_k)$ and all the x_k are generated by T. An *instance* is a finite set of function-free facts. The terms appearing in an instance (resp. atom) are denoted by $\mathit{terms}(I)$ (resp. $\mathit{terms}(\alpha)$).

Existential Rules An *existential rule* (or just *rule*) takes the form:

$$\forall \mathbf{x} \forall \mathbf{z}.[\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y}.\psi(\mathbf{x}, \mathbf{y})],$$

where $\varphi(\mathbf{x}, \mathbf{z})$ and $\psi(\mathbf{x}, \mathbf{y})$ are non-empty conjunctions of function-free atoms, and tuples of variables $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are pairwise disjoint. We call φ the *body* and ψ the *head* of the rule. For brevity, quantifiers are often omitted.

We frequently use *Skolemisation* to interpret rules in *Herbrand* interpretations, which are defined as possibly infinite sets of facts. In particular, for each rule ρ and each variable $y_i \in \mathbf{y}$, let f_ρ^i be a function symbol globally unique for ρ and y_i of arity $|\mathbf{x}|$; furthermore, let θ_{sk} be the substitution such that $\theta_{sk}(y_i) = f_\rho^i(\mathbf{x})$ for each $y_i \in \mathbf{y}$. Then, the Skolemisation $\mathit{sk}(\rho)$ of ρ is the following rule: $\varphi(\mathbf{x}, \mathbf{z}) \to \psi(\mathbf{x}, \mathbf{y})\theta_{sk}$.

A *linear rule* is an existential rule whose body is restricted to a single atom. For ease of presentation, we will consider only rules without any constants. As usual, we also assume that rules have only a single atom in the head. This can be done without loss of generality.

Skolem Chase The *chase* [16,14] (or canonical model) is a classical tool in OMQA. In this paper, we use the *Skolem chase* variant ([17]). Let $\rho = \varphi \to \psi$ be a Skolemised rule, and let I be a set of facts. A set of facts S is a consequence of ρ on I if a substitution σ exists that maps the variables in ρ to the terms occurring in I (denoted by $\mathit{terms}(I)$) such that $\varphi\sigma \subseteq I$ and $S \subseteq \psi\sigma$. The *result* of applying ρ to I, written $\rho(I)$, is the union of all consequences of ρ on I. If Ω is a set of Skolemised rules, we set $\Omega(I) = \bigcup_{\rho \in \Omega} \rho(I)$. Let I be a finite set of facts, let $\mathcal{R}$ be a set of rules, let $\mathcal{R}' = \mathit{sk}(\mathcal{R})$, and let $\mathcal{R}'_f$ and $\mathcal{R}'_n$ be the subsets of $\mathcal{R}'$ containing rules with and without function symbols, respectively. The *chase sequence* for I and $\mathcal{R}$ is a sequence of sets of facts $I^0_\mathcal{R}, I^1_\mathcal{R}, \ldots$, where $I^0_\mathcal{R} = I$ and for each $i > 0$, set $I^i_\mathcal{R}$ is defined as follows:

- if $\mathcal{R}'_n(I^{i-1}_\mathcal{R}) \not\subseteq I^{i-1}_\mathcal{R}$, then $I^i_\mathcal{R} = I^{i-1}_\mathcal{R} \cup \mathcal{R}'_n(I^{i-1}_\mathcal{R})$
- otherwise $I^i_\mathcal{R} = I^{i-1}_\mathcal{R} \cup \mathcal{R}'_f(I^{i-1}_\mathcal{R})$

The *chase* of I and $\mathcal{R}$, written $\mathit{chase}(I, \mathcal{R})$, is defined as $\bigcup_i I^i_\mathcal{R}$; note that $\mathit{chase}(I, \mathcal{R})$ can be infinite. However, the chase has a simple structure when linear rules are considered: each atom can be "chased" independently.

Property 1 (Decomposition of the chase). Let $\mathcal{R}$ be a set of linear rules and I be an instance. It holds that:

$$\mathit{chase}(I, \mathcal{R}) = \cup_{\alpha \in I}\ \mathit{chase}(\{\alpha\}, \mathcal{R})$$

Regular Languages A regular language can be represented either by a regular expression or by a non-deterministic finite automaton (NFA). Let Σ be a finite set of symbols. A regular expression over Σ is defined by the grammar: $\mathcal{E} \to \varepsilon \mid a \mid \mathcal{E} \cdot \mathcal{E} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E}^*$, where $a \in \Sigma$ and ε denotes the empty word. We use $L(\mathcal{E})$ to denote the language defined by $\mathcal{E}$. An NFA over Σ is a tuple $\mathbb{A} = (S, \Sigma, \delta, s_0, F)$, where S is a finite set of states, $\delta \subseteq S \times \Sigma \times S$ is the transition relation, $s_0 \in S$ is the initial state and $F \subseteq S$ is the set of final states. If $\mathbb{A}$ is an automaton and s and s' are two states of $\mathbb{A}$, we denote by $\mathcal{L}_\mathbb{A}(s, s')$ the set of words w for which there is path from s to s' in $\mathbb{A}$ labeled by w.

Regular Path Queries Let $\mathcal{P}$ be a set of predicates. Let us define $\mathcal{P}_2^\pm = \mathcal{P}_2 \cup \{r^- \mid r \in \mathcal{P}_2\}$ and $\mathcal{P}_r = \mathcal{P}_2^\pm \cup \mathcal{P}_1$, where $\mathcal{P}_i$ ($i \in \{1, 2\}$) denotes the predicates of arity i. A *two-way regular path query* (RPQ[3]) is a query of the form $q(x, x') = \mathcal{E}(x, x')$, where $\mathcal{E}$ is a regular expression defining a language over $\mathcal{P}_r$.

[3] As we only consider the two-way variant, we will use the abbreviation RPQ instead of the more traditional 2RPQ.

Given an interpretation $\mathcal{I}$, a *path* from a_0 to a_n in $\mathcal{I}$ is a sequence $a_0 r_1 a_1 r_2 \dots r_n a_n$ such that for any i such that $1 \leq i \leq n$, a_i is an element of the domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$, every r_i is a symbol from $\mathcal{P}_r$ and:

– if $r_i = a \in \mathcal{P}_1$, then $a_i = a_{i-1} \in a^{\mathcal{I}}$;
– if $r_i \in \mathcal{P}_2$, then $(a_{i-1}, a_i) \in r_i^{\mathcal{I}}$;
– if $r_i = r^-$ with $r \in \mathcal{P}_2$, then $(a_i, a_{i-1}) \in r^{\mathcal{I}}$.

The *label* $\lambda(p)$ of path $p = a_0 r_1 a_1 r_2 \dots r_n a_n$ is the word $r_1 r_2 \dots r_n$. For any language L over $\mathcal{P}_r$, the semantics of L with respect to an interpretation $\mathcal{I}$ is defined by:

$$L^{\mathcal{I}} = \{(a_0, a_n) \mid \text{there is some path } p \text{ from } a_0 \text{ to } a_n \text{ such that } \lambda(p) \in L\}.$$

A *match* for an RPQ $q(x, x') = \mathcal{E}(x, x')$ in an interpretation $\mathcal{I}$ is a mapping π from the variables of q to elements of $\Delta^{\mathcal{I}}$ such that $(\pi(x), \pi(x')) \in L(\mathcal{E})^{\mathcal{I}}$.

A *certain answer* to $q(x_1, x_2)$ with respect to $(I, \mathcal{R})$ is a pair of constants (a_1, a_2) such that for every model $\mathcal{I}$ of $(I, \mathcal{R})$, there is a match π for q such that $\pi(x_1) = a_1^{\mathcal{I}}$ and $\pi(x_2) = a_2^{\mathcal{I}}$. As matches are preserved under homomorphisms, it holds that (a_1, a_2) is a certain answer to $q(x_1, x_2)$ w.r.t. $(I, \mathcal{R})$ if and only if there is a match for $(a_1^{\mathcal{I}}, a_2^{\mathcal{I}})$ in $\mathcal{I} = \mathit{chase}(I, \mathcal{R})$. The *RPQ Answering problem* asks, given an RPQ $q(x_1, x_2)$, an instance I, a set of existential rules $\mathcal{R}$, and two constants $(a_1, a_2) \in \mathit{terms}(\mathcal{I}) \times \mathit{terms}(\mathcal{I})$, whether (a_1, a_2) is a certain answer to $q(x_1, x_2)$.

Computational Complexity and Turing Machines We assume the reader to be familiar with standard complexity classes. In particular, we will consider P, NP, PSPACE, APSPACE (alternating PSPACE), and EXPTIME. We recall that APSPACE = EXPTIME.

To fix notations, we recall that an *alternating Turing machine (TM)* is given by a 5-tuple $\mathcal{M} = (Q, \Gamma, \delta, q_0, g)$ where:

– Q is the finite set of states;
– Γ is the finite tape alphabet;
– $\delta : Q \times \Gamma \to (Q \times \Gamma \times \{L, R\})^2$ is the transition function;
– $q_0 \in Q$ is the initial state;
– $g : Q \to \{\wedge, \vee, \text{accept}, \text{reject}\}$ specifies the type of each state.

Note that without loss of generality, we consider TMs having the following properties:

– for every universal ($\wedge$) or existential ($\vee$) configuration, there exist exactly two applicable transitions;
– the machine directly accepts any configuration whose state s is such that $g(s) =$ accept;
– the TM never tries to go to the left of the initial position.

We say $\mathcal{M}$ is *polynomially space-bounded* ($\mathcal{M}$ is a PSPACE TM) if there exists a polynomial p such that on input x, $\mathcal{M}$ visits only the first $p(|x|)$ tape cells. We assume w.l.o.g. that the alternating PSPACE TMs we consider terminate on every input.

3 Evaluating Regular Path Queries over Linear Rules

We consider the problem of computing the certain answers to a regular path query and show how to adapt the construction in [5] to the case of linear rules. There are two main ingredients in the original algorithm for DL-Lite:

– a path in the chase is guessed step by step, keeping in memory only the current constant of the instance and current state of the automaton;
– when a path goes through the Skolem part of the chase, these constants are not guessed, but the state in which the automaton is when the path returns to constants of the instance is guessed, thanks to a precomputed table.

3.1 Additional Challenges with Linear Rules

There are two main differences between DL-Lite and linear rules that need to be handled. First, in DL-Lite, it is enough to know the predicate of the atom in which an constant has been created during the chase and the position at which it appeared in that atom to determine all the atoms that contain that constant in the chase. This is not true if we consider general linear rules, as illustrated by the following example:

Example 1 (More complex types are needed). Let us consider the following rules:

$$h(x,y,z) \to h(z,x,y) \qquad h(x,x,y) \to q(y)$$

and instance $I = \{h(a,b,b), h(c,d,e)\}$. Observe that while a and c occur in the same position of atoms with the same predicate, $q(a)$ is in $\mathsf{chase}(I,\mathcal{R})$, while $q(c)$ is not.

Second, the following looping property is central to the algorithm from [5].

Definition 1 (Looping property). *An ontology $\mathcal{R}$ fulfills the looping property if it holds that for any instance I, for any path $a_0 r_1 a_1 \ldots r_n a_n$ in $\mathsf{chase}(I,\mathcal{R})$ such that* (i) *a_i and a_{i+1} are Skolem terms,* (ii) *a_i is a subterm of a_{i+1}, and* (iii) *a_1 and a_n are original constants, there exists $k \geq i$ such that $a_k = a_i$.*

Indeed, DL-Lite$_{\mathcal{R}}$ fulfills the looping property (as do many other DLs). However, linear rules do not, as is witnessed by Example 2.

Example 2 (Failure of looping property). Consider the instance $I_e = \{t(a,b)\}$ and the ruleset $\mathcal{R}_e$ consisting of the following rules:

$$\begin{array}{ll} t(x,y) \to r(y,z) & q(x,y,z) \to p(y,z) \\ r(x,y) \to q(x,y,z) & q(x,y,z) \to p(z,x) \end{array}$$

The chase for I_e and $\mathcal{R}_e$ contains the following atoms:

$$r(b,f_1(b)) \quad q(b,f_1(b),f_2(b,f_1(b))) \quad p(f_1(b),f_2(b,f_1(b))) \quad p(f_2(b,f_1(b)),b)$$

There is thus a path $b\, r\, f_1(b)\, p\, f_2(b,f_1(b))\, p\ b$ going from the initial constant b to b, that passes by $f_1(b)$ but does not return via $f_1(b)$.

3.2 Adapting the DL-Lite$_\mathcal{R}$ algorithm

To take care of the first difficulty, we utilize a finer notion of type, which has similar properties to the one used in [5].

Definition 2 (Type). *A* type *is a pair $(r, \mathcal{P})$ where r is a predicate of arity k and $\mathcal{P}$ is a partition of $\{1, \ldots, k\}$.*

With each atom, we can associate a type, representing the way terms are repeated in the atom.

Definition 3 (Type of an atom). *Let α be an atom, whose arity is k. The type of α is the pair $(r, \mathcal{P})$ where p is the predicate of α and $\mathcal{P}$ is the partition of $\{1, \ldots, k\}$ such that i and j belong to the same partition iff the i^{th} and the j^{th} arguments of α are equal.*

Note that if two atoms α_1 and α_2 are of same type, there exists an injective substitution θ_{12} such that $\alpha_2 = \alpha_1\theta_{12}$.

Property 2. Let I be an instance, and $\mathcal{R}$ be a set of linear rules. Let α_1 and α_2 be two atoms of I of same type and θ_{12} such that $\alpha_2 = \alpha_1\theta_{12}$. Then for every atom β such that $\beta \in \mathit{chase}(\{\alpha_1\}, \mathcal{R})$, $\beta\theta_{12} \in \mathit{chase}(\{\alpha_2\}, \mathcal{R})$.

Let us define for any atom $\alpha \in \mathit{chase}(I, \mathcal{R})$, the restriction of $\mathit{chase}(I, \mathcal{R})$ to α, denoted $\mathit{chase}(I, \mathcal{R})_{|\alpha}$, as the subset of $\mathit{chase}(I, \mathcal{R})$ consisting of those atoms whose terms are generated by $\mathit{terms}(\alpha)$. Observe that by the preceding property, if $\mathit{type}(\alpha) = \mathit{type}(\beta)$, then $\mathit{chase}(I, \mathcal{R})_{|\alpha}$ is isomorphic to $\mathit{chase}(\{\beta\}, \mathcal{R})$.

We can overcome the second difficulty by generalizing the `Loop` table introduced in [5], which keeps track of the paths that occur 'below' a given type. Intuitively, a type T is in the cell indexed by (s_i, j, s'_i, j') if and only if below any atom of type T, there is a path going from the term in position j to the term in position j' labeled by a word that takes $\mathbb{A}$ from state s_i to state s'_i.

Definition 4 (`Loop`). *Let $\mathcal{R}$ be a set of linear rules and $\mathbb{A}$ be an NFA. A `Loop` table has cells indexed by tuples $(s_i, j, s_{i'}, j')$ such that s_i and $s_{i'}$ are states of $\mathbb{A}$ and j and j' are integers between 1 and w, where w is the maximum arity appearing in the ruleset. Cells contain types. A `Loop` table is:*

- sound *if for every $T \in (s_i, j, s_{i'}, j')$ it holds that for every atom α of type T appearing in some* chase$(\{\alpha'\}, \mathcal{R})$ *(with the predicate of α' appearing in $\mathcal{R}$), there is a path p in the restriction of* chase$(I, \mathcal{R})$ *to α that goes from argument j of α to argument j' of α such that $\lambda(p) \in \mathcal{L}_\mathbb{A}(s_i, s_{i'})$.*
- complete *if for every atom α of type T (whose predicate appears in $\mathcal{R}$), if there is path p from argument j to argument j' of α in* chase$(\{\alpha\}, \mathcal{R})$ *such that $\lambda(p) \in \mathcal{L}_\mathbb{A}(s_i, s_{i'})$, then $T \in (s_i, j, s_{i'}, j')$.*

It is direct from the definition that there exists a unique sound and complete `Loop` table, and in what follows, we use `Loop` to denote this table.

The table `Loop` can be constructed using Algorithm 1. Line 5 initializes the table by stating than one can go from a position to the same position without reading any word (and thus not moving in the automaton). Lines 8 and 10 correspond to going through a single edge, reading its label either as an r or an r^-, in the case where both terms are distinct. Lines 13 to 16 do the same thing when both arguments are equal. Line 19 deals with unary predicates. Finally, Lines 23 and 26 saturate the table through respectively transitive closure and propagation of paths from a child to its parent.

Algorithm 1: Creating the `Loop` table

```
Data: A set of linear rules $\mathcal{R}$
Result: A sound and complete Loop table
/* Initialization step                                              */
1  foreach arity $k$ do
2      foreach type $T$ of predicate of arity $k$ do
3          for $j \in \{1, \dots, k\}$ do
4              for $s_i \in Q(\mathbb{A})$ do
5                  $\mathtt{Loop}(s_i, j, s_i, j) \leftarrow \mathtt{Loop}(s_i, j, s_j, j) \cup \{T\}$;
6  for type $T$ based on $r(x, y)$ do
7      if $s_2 \in \delta(s_1, r)$ then
8          $\mathtt{Loop}(s_1, 1, s_2, 2) \leftarrow \mathtt{Loop}(s_1, 1, s_2, 2) \cup \{T\}$;
9      if $s_2 \in \delta(s_1, r^-)$ then
10         $\mathtt{Loop}(s_1, 2, s_2, 1) \leftarrow \mathtt{Loop}(s_1, 2, s_2, 1) \cup \{T\}$;
11 for type $T$ based on $r(x, x)$ do
12     if $s_2 \in \delta(s_1, r) \cup \delta(s_1, r^-)$ then
13         $\mathtt{Loop}(s_1, 1, s_2, 1) \leftarrow \mathtt{Loop}(s_1, 1, s_2, 1) \cup \{T\}$;
14         $\mathtt{Loop}(s_1, 1, s_2, 2) \leftarrow \mathtt{Loop}(s_1, 1, s_2, 2) \cup \{T\}$;
15         $\mathtt{Loop}(s_1, 2, s_2, 1) \leftarrow \mathtt{Loop}(s_1, 2, s_2, 1) \cup \{T\}$;
16         $\mathtt{Loop}(s_1, 2, s_2, 2) \leftarrow \mathtt{Loop}(s_1, 2, s_2, 2) \cup \{T\}$;
17 for type $T$ based on $a(x)$ do
18     if $s_2 \in \delta(s_1, a)$ then
19         $\mathtt{Loop}(s_1, 1, s_2, 1) \leftarrow \mathtt{Loop}(s_1, 1, s_2, 1) \cup \{T\}$;
/* Saturation step                                                  */
20 while something added do
21     for $T$ a type do
22         if $T \in Loop(s_1, j_1, s_2, j_2) \cap Loop(s_2, j_2, s_3, j_3)$ then
23             $\mathtt{Loop}(s_1, j_1, s_3, j_3) \leftarrow \mathtt{Loop}(s_1, j_1, s_3, j_3) \cup \{T\}$;
24     for $\alpha \to \beta \in \mathcal{R}$, of respective types $T_\alpha, T_\beta$ do
25         if the same variable appears in $\alpha$ at $i_\alpha$ and $\beta$ at $i_\beta$ (resp. $j_\alpha$ and $j_\beta$),
           $T_\beta \in Loop(s_1, i_\beta, s_2, j_\beta)$ then
26             $\mathtt{Loop}(s_1, i_\alpha, s_2, j_\alpha) \leftarrow \mathtt{Loop}(s_1, i_\alpha, s_2, j_\alpha) \cup \{T_\alpha\}$;
```

Property 3. Let $\mathcal{R}$ be a set of linear rules, I be an instance and $\alpha \in I$. The following are equivalent:

1. *type*$(\alpha) \in \mathtt{Loop}(s, i, s', j)$

2. there is a path $p = a_0 r_1 a_1 \ldots r_n a_n$ in *chase*$(I, \mathcal{R})_{|\alpha}$ with a_0 appearing at position i in α, a_n appearing at position j in α, and $\lambda(p) \in \mathcal{L}_{\mathbb{A}}(s, s')$.

Proof. ($\Rightarrow$) We prove, by induction on the order of addition of types that whenever a type is added to a cell in $\texttt{Loop}(s, i, s', j)$, the second condition is fulfilled as well. If *type*(α) is added to $\texttt{Loop}(s_i, j, s_i, j)$ at Line 5, the empty word defines a trivial path from any position existing in α to itself, and takes the automaton from any state to itself. If *type*(α) is added to $\texttt{Loop}(s_1, 1, s_2, 2)$ at Line 8, α is a binary atom of the form $r(e_1, e_2)$, and there is indeed a path from e_1 to e_2 labeled r. Moreover, there is a transition in $\mathbb{A}$ from s_1 to s_2 labeled by r, which concludes this case. The reasoning is similar for types added via Line 10 and Lines 13 to 16. If *type*(α) is added at Line 23, it must have already been added to $\texttt{Loop}(s_1, j_1, s_2, j_2)$ and $\texttt{Loop}(s_2, j_2, s_3, j_3)$. By the induction assumption, there is a word w_1 (resp. w_2) in $\mathcal{L}_{\mathbb{A}}(s_1, s_2)$ (resp. $\mathcal{L}_{\mathbb{A}}(s_2, s_3)$) that labels a path from the position j_1 (resp. j_2) of an atom α of type T to the position j_2 (resp. j_3). Thus $w_1 \cdot w_2$ labels a path from position j_1 in α to position j_3 in α and belongs to $\mathcal{L}_{\mathbb{A}}(s_1, s_3)$. Finally, let us assume that *type*(α) is added to $\texttt{Loop}(s_1, i_{\alpha'}, s_2, j_{\alpha'})$ at Line 26. By assumption, there is a rule $\alpha' \to \beta'$ in $\mathcal{R}$ such that α and α' have the same type, *type*(β') is in $\texttt{Loop}(s_1, i_{\beta'}, s_2, j_{\beta'})$, and the same variable appears at position $i_{\alpha'}$ (resp. $j_{\alpha'}$) in α' and $i_{\beta'}$ (res. $j_{\beta'}$) in β'. By the induction assumption, there is a word $w \in \mathcal{L}_{\mathbb{A}}(s_1, s_2)$ that labels a path from $i_{\beta'}$ to $j_{\beta'}$. Now, let us observe that any two terms that are at positions $i_{\alpha'}$ and $j_{\alpha'}$ of the same atom of type *type*(α') are also at position $i_{\beta'}$ and $j_{\beta'}$ of an atom of type *type*(β') in *chase*$(D, \mathcal{R})_{|\alpha}$ because it is a model of $\alpha' \to \beta'$. Thus, w is also the label of a path from the term at position i'_α to the term at position j'_α, which concludes the proof.

($\Leftarrow$) We suppose that the second statement holds and reason by induction on the length n of the path $p = a_0 r_1 a_1 \ldots r_n a_n$.

Base case, path of length 0: both states and database constants are thus equal, and the type is added by the initialization in Line 5.

Base case, path of length 1: $\alpha' = r_1(a_0, a_1)$ belongs to *chase*$(I, \mathcal{R})_{|\alpha}$, and $r_1 \in \mathcal{L}_{\mathbb{A}}(s, s')$. If $a_0 \neq a_1$, then *type*(α') is added to the cells $(s, 1, s', 2)$ and $(s, 1, s', 2)$ in Lines 8 and 10. If $a_0 = a_1$, then *type*(α') is added to the four cells (s, i', s', j') with $i', j' \in \{1, 2\}$ (Lines 13-16). As α' belongs to *chase*$(I, \mathcal{R})_{|\alpha}$, there exists a finite sequence of atoms $\alpha = \alpha_0, \ldots, \alpha_m = \alpha'$ such that α_{i+1} belongs to $\rho_i(\alpha_i)$ for some rule $\rho_i \in \mathcal{R}$. By using m applications of Line 26, we obtain *type*$(\alpha) \in \texttt{Loop}(s, i, s', j)$.

Induction step: let us assume that the result holds for any path of length up to $n-1$, $n \geq 2$, and consider the path $p = a_0 r_1 a_1 \ldots r_n a_n$. First consider the case in which a_k is contained in α for some $1 \leq k < n$, and let l be a position of a_k in α. There exists a path from a_0 to a_k of length strictly smaller than n, and similarly from a_k to a_n. By the induction assumption, *type*(α) is in both $\texttt{Loop}(s, i, s'', l)$ and $\texttt{Loop}(s'', l, s', j)$ for some state s''. An application of Line 23 yields *type*$(\alpha) \in \texttt{Loop}(s, i, s', j)$. Next suppose there is no a_k ($1 \leq k < n$) that occurs in α, and let β be the atom in which a_1 is created (at position k'). This atom is well defined as we consider rules with atomic head. We know that a_0 (resp. a_n) must occur in β, let us say at position i' (resp. j'). Indeed, if it was not the case, α should contain a term among $a_1, \ldots, a_{n-1}$ which contradicts our earlier assumption. By the induction hypothesis, *type*(β) belongs to $\texttt{Loop}(s, i', s'', k')$ and to $\texttt{Loop}(s'', k', s', j')$ for some state s''. Hence, by Line 23, *type*(β) is in the cell $\texttt{Loop}(s, i', s', j')$. By (repeated) application of Line 26, *type*(α) is in the cell $\texttt{Loop}(s, i, s', j)$, which concludes the proof. □

Property 4. Algorithm 1 runs in exponential time, and in polynomial time if the predicate arity is bounded.

Proof. There are polynomially many cells in the table, each of which can contain at most all types. The number n_t of distinct types is single exponential (and polynomial for bounded-arity predicates). The first for loop runs in $\mathcal{O}(n_t)$, the next two run in polynomial time, and the while loop is performed at most n_t times. □

The remainder of the decision procedure is very close to the original algorithm for DL-Lite$_{\mathcal{R}}$, but we recall it here (Algorithm 2) in the interest of self-containment. The idea is as follows: starting from a constant a and the initial state of $\mathbb{A}$, we guess the next constant in I on a path from a to b and the state of $\mathbb{A}$ after taking this step (Line 7). We then check that this choice is valid, i.e., there is indeed a path from a to the guessed constant which takes the automaton from the initial state to the current guessed state. This can be done either by a checking that a corresponding unary or binary atom is entailed (Lines 9 and 10), or by checking that a path going through the Skolem part of the chase allows us to reach the next constant in the required state, using the $\texttt{Loop}$ table (Lines 12 to 14). We repeat this procedure until we reach the constant b in a final state, or hit the maximal path length. Note that at Line 12, α is uniquely defined if it exists (it may not exist e.g., if c and d are different but are at positions that should have identical terms according to T).

The following property will be used to establish correctness of the algorithm.

Property 5. At the beginning of each iteration of the while loop of Algorithm 2, it holds that there is a path from a to the first element of $\texttt{current}$ that takes the NFA $\mathbb{A}$ from the initial state s_0 to the state in the second argument of $\texttt{current}$.

Proof. At the beginning of the first iteration of the while loop, $\texttt{current}$ is equal to (a, s_0). Thus, the path a, whose label is ε, goes from a to a and $\varepsilon \in \mathcal{L}_{\mathbb{A}}(s_0, s_0)$.

Let (a_i, s_i) be the content of $\texttt{current}$ at the beginning of the i^{th} iteration of the while loop. Let w_i be the label of a path from a_0 to a_i such that $w_i \in \mathcal{L}_{\mathbb{A}}(s_0, s_i)$. If there is an $(i+1)^{\text{th}}$ iteration, either (s, σ, s') or (T, i_c, i_d) has been guessed, and the corresponding check was successful. Let us consider each case:

– if (s, σ, s') has been guessed and checked, we have two cases:
  - $\sigma \in \mathcal{P}_2^{\pm}$, and there is a path from a_i to a_{i+1} in *chase*$(I, \mathcal{R})$ labeled by σ. Moreover, σ labels an edge from s to s' in $\mathbb{A}$. We can thus define $w_{i+1} = w_i.\sigma$
  - $\sigma = A$, and $I, \mathcal{R} \models A(c)$. As $c = d$, we can again define $w_{i+1} = w_i.\sigma$

– if (T, i_c, i_d) has been guessed, it means that T belongs to $\texttt{Loop}(s_i, i_c, s_{i+1}, i_d)$. By the definition of $\texttt{Loop}$, there is a path p (in the Skolem part) from any term at position i_c of an atom of type T to the position i_d of an atom of type T such that $\lambda(p) \in \mathcal{L}_{\mathbb{A}}(s, s')$. Let α be as defined Line 12. As $I, \mathcal{R} \models \alpha$, where *type*$(\alpha) = T$, a_i appears at position i_c of α, and a_{i+1} appears at position i_d of α, there is such a path from a_i to a_{i+1}. We can thus set $w_{i+1} = w_i.p$. □

Algorithm 2: RPQ answering over linear rules

```
Input: An NFA $\mathbb{A}$, an instance $I$, a set of linear rules $\mathcal{R}$, $(a, b) \in \mathit{terms}(I) \times \mathit{terms}(I)$
Output: Yes if and only if $(a, b)$ is a certain answer to the query $q$ defined by $\mathbb{A}$
1  if $(I, \mathcal{R})$ is not satisfiable then
2      return Yes
3  current = $(a, s_0)$;
4  count = $0$, max = $|\mathbb{A}| \times |I|$;
5  while $count < max$ and $current \notin \{(b, s_f) \mid s_f \in F\}$ do
6      Define $(c, s) =$ current;
7      Guess $(d, s')$ together with $(s, \sigma, s') \in \delta$ or $T, i_c, i_d$ such that
       $T \in \mathtt{Loop}(s, i_c, s', i_d)$;
8      if $(s, \sigma, s')$ was guessed then
9          if $\sigma \in \mathcal{P}_2^{\pm} \wedge (I, \mathcal{R} \not\models \sigma(c, d))$ then return No;
10         if $\sigma = A \wedge (c \neq d \vee I, \mathcal{R} \not\models A(c))$ then return No;
11     if $T, i_c, i_d$ was guessed then
12         Let $\alpha$ be of type $T$ such that $c$ is at position $i_c$ and $d$ is at position $i_d$; other terms
           are set to fresh variables
13         if $\alpha$ does not exist then return No;
14         if $I, \mathcal{R} \not\models \alpha$ then return No;
15     current = $(d, s')$, count = count $+1$;
16 if current$= (b, s_f)$ for some $s_f \in F$ then return Yes else return No;
```

Property 6. There is an execution of Algorithm 2 that outputs Yes iff the RPQ given by $\mathbb{A}$ is entailed from $(I, \mathcal{R})$.

Proof. ($\Rightarrow$) If the algorithm outputs Yes, the while loop has been exited with current equal to (b, s_f), with s_f a final state of $\mathbb{A}$. By Property 5, this means that there is a path from a to b whose label takes $\mathbb{A}$ from s_0 to s_f, hence is accepted by $\mathbb{A}$. This show that whenever Algorithm 2 accepts, (a, b) is a certain answer to the RPQ given by $\mathbb{A}$.

($\Leftarrow$) If (a, b) is a certain answer to the RPQ based upon $\mathbb{A}$, then there is path of minimal length $p = a'_0 r_1 a'_1 \dots r_n a'_n$ from $a = a'_0$ to $b = a'_n$ in $\mathit{chase}(I, \mathcal{R})$ such that $\lambda(p) = r_1 \dots r_n \in \mathcal{L}_{\mathbb{A}}(s_0, s_f)$ for some final state s_f. Let $s'_0 s'_1 \dots s'_n$ be a sequence of states of $\mathbb{A}$ such that s'_n is a final state of $\mathbb{A}$ and for every $1 \leq i \leq n$, $(s_{i-1}, r_i, s_i) \in \delta$. Since p is of minimal length, there is no pair (i, j) with $i \neq j$ such that $(a_i, s_i) = (a_j, s_j)$. Let us consider the sequence $p' = ((a_i, s_i))_i$ such that:

- for any i, a_i is the i^{th} constant, say a'_{k_i}, in p belonging to $\mathit{terms}(I)$;
- for any i, $s_i = s'_{k_i}$.

Moreover, for any i, if $k_{i+1} = k_i + 1$, we define $\mathit{aux}_i = (s_i, r_{i+1}, s_{i+1})$. Otherwise, let $\mathit{aux}_i = (\mathit{type}(\alpha), i_c, i_d)$,where:

- α is such that $\alpha \in I$ and $\mathit{type}(\alpha) \in \mathtt{Loop}(s_i, i_c, s_{i+1}, i_d)$;
- a_{k_i} appears at position i_c of α and $a_{k_{i+1}}$ appears at position i_d of α.

In the second case, it is possible to define aux_i in such a way, as the path $p_s = a'_{k_i} r_{k_i+1} \dots a'_{k_{i+1}}$ goes from a_{k_i} to $a_{k_{i+1}}$ and belongs to $\mathcal{L}_{\mathbb{A}}(s_i, s_{i+1})$ by definition of s_i. We show that the sequence of guesses $(a_i, s_i, \mathit{aux}_i)$ leads Algorithm 2 to accept. Since p is minimal, the length of p' is less than $|\mathbb{A}| \times |I|$. Moreover, $a_n = b$ and s_f is a final state. Thus, the only way for Algorithm 2 to reject with this sequence of guesses is to reject during checks, *i.e.*, one of the checks performed at Lines 9, 10, 12 or 14 fails. Let $(a_i, s_i, \mathit{aux}_i)$ be the guess at one of the steps. If aux_i is of the form (s_i, r_{i+1}, s_{i+1}), then a_{k_i} and $a_{k_{i+1}}$ are consecutive elements in p, and there is an atom $r_{i+1}(a_{k_i}, a_{k_{i+1}})$ in $\mathit{chase}(I, \mathcal{R})$. Thus, $r_{i+1}(a_{k_i}, a_{k_{i+1}})$ is entailed by I and $\mathcal{R}$, and the check at Line 9 or 10 (depending on r_{i+1} being a binary or unary atom) is successful. If aux_i is of the form $(\mathit{type}(\alpha), i_c, i_d)$, then there is $\alpha \in I$ such that $\mathit{type}(\alpha) \in \texttt{Loop}(s_i, i_c, s_{i+1}, i_d)$, and with a_{k_i} (resp. $a_{k_{i+1}}$) appearing at position i_c (resp. i_d) of α. The atom α fulfills the conditions of Lines 12 and 14. Thus the defined sequence never triggers a rejection from Algorithm 2, which concludes the proof. $\square$

Theorem 1. RPQ *Answering in the presence of linear existential rules is:*

- *in* NL *in data complexity*
- *in* PTIME *in combined complexity with bounded arity*
- *in* EXPTIME *in combined complexity with unbounded arity*

Proof. Algorithm 2 is a non-deterministic algorithm that needs to keep in memory the current state, the current constant, and the number of iterations done so far. It performs two types of operations: entailment checks and accessing the contents of the `Loop` table (more precisely, deciding whether $T \in \texttt{Loop}(s, i_c, s', i_d)$). Hence, it can be seen as an NL algorithm making oracle calls whenever an entailment check is performed or a cell of `Loop` is retrieved. Entailment checks are in NL in data complexity, and `Loop` is independent from the data: the overall algorithm thus runs in NL in data complexity. In combined complexity with bounded arity, entailment checks can be performed in PTIME, while `Loop` can be computed in polynomial time: the overall algorithm is thus in PTIME with bounded arity. In the unbounded arity case, the entailment checks can be performed in PSPACE, while the `Loop` table can be computed in EXPTIME: the algorithm thus runs in EXPTIME. $\square$

4 Lower Bound

It is already known that the data complexity (resp. combined complexity) of RPQs under linear rules (resp. linear rules with bounded arity) is NL-hard (resp. PTIME-hard) [5], which matches the upper bounds obtained in the preceding section. We thus focus on providing a matching EXPTIME lower bound for the combined complexity of evaluating RPQs under linear rules of unbounded arity. The proof is done by simulating an alternating PSPACE TM. It is already known that PSPACE TMs can be simulated by means of linear rules [12]. In the following, we explain how to adapt this construction to simulate alternating TMs. Note that in this section, we will use rules with multiple atoms in the head: this is done to simplify the presentation, and a classical transformation allows us to get the same lower bound for rules with atomic heads.

The intuition is as follows: the construction in [12] represents the configuration of a TM $\mathcal{M}$ by a single atom of polynomial arity. The initial configuration can thus be represented by an instance $I_{\mathcal{M}}$ containing a single atom. Then, for each transition of the TM, polynomially many linear rules are created, each one representing the action of the transition on a cell at a given position. All these rules are part of $\mathcal{R}_{\mathcal{M}}$. The initial configuration of the TM is accepted if and only if an atom encoding a configuration having an accepting state is entailed by $I_{\mathcal{M}}$ and $\mathcal{R}_{\mathcal{M}}$.

We modify this construction in the following way to deal with alternating Turing machines: to each atom, we add two positions, that will act as "input" and "output" positions. Moreover, we will maintain the following property: there is a path, whose edges are all labeled by the same predicate p, from the input position of α to the output position of α entailed by $\mathit{chase}(I_{\{\alpha\}}, \mathcal{R}_{\mathcal{M}})$ if and only if the configuration represented by α is accepted by $\mathcal{M}$. This is true in the following cases:

- the state of the current configuration is accepting. It is then enough to add a p-edge from i_c to o_c; this is possible as the Turing machine is assumed to never leave an accepting state;
- the current state is existential and one of the two successor configurations is accepting: we thus add p-edges from the input of the current configuration to the input of the two children, and from the output of the two children to the output of the current configuration;
- the current state is universal, and both successor configurations are accepting: we thus add p-edges from the input of the current configuration to the input of the first successor configuration, then from the output of that configuration to the input of the other successor, and lastly from the output of the second successor to the output of the current configuration.

We now formalize the construction sketched above, staying as close as possible to the notations in [12].

Turing Machine Given an alternating PSPACE TM and an input x, we can represent a configuration c reached during the computation by storing the content of the first $p(|x|)$ cells, as well as the position of the head of the tape and the current state of the TM. Adding input and output positions, this can be encoded by a predicate *conf* of arity $2p(|x|) + 3$:

$$\mathit{conf}(i_c, state, cell_1, cur_1, cell_2, cur_2, \ldots, cell_{p(|x|)}, cur_{p(|x|)}, o_c),$$

where $state$ contains the state identifier, $cell_i$ represents the content of the i^{th} cell, cur_i is equal to 1 if the head of the Turing machine is on cell i and 0 otherwise, and i_c and o_c are the input and output terms of this atom. We say that the above atom *represents* configuration c. Given an atom α, the term at its input (resp. output) position is denoted by $i(\alpha)$ (resp. $o(\alpha)$). We denote by $I_{\mathcal{M},x}$ the instance containing a single atom representing the initial configuration of $\mathcal{M}$ on input x.

For every state q_f with $g(q_f) =$ accept, we create the following rule:

$$\mathit{conf}(i_c, q_f, \ldots, o_c) \to p(i_c, o_c). \tag{1}$$

For each transition $\delta(q,\gamma) = \{(q',\gamma',L),(q'',\gamma'',L)\}$ such that $g(q) = \vee$, we create the rule

$$\begin{aligned}&\mathit{conf}(i_c, q, cell_1, cur_1, \ldots, cell_{i-1}, 0, \gamma, 1, \ldots, o_c) \to \\ &\quad \exists i_{c'}, o_{c'}, i_{c''}, o_{c''}\ \mathit{conf}(i_{c'}, q', cell_1, cur_1, \ldots, cell_{i-1}, 1, \gamma', 0, \ldots, o_{c'}), \\ &\qquad \mathit{conf}(i_{c''}, q'', cell_1, cur_1, \ldots, cell_{i-1}, 1, \gamma', 0, \ldots, o_{c''}), \\ &\qquad\qquad p(i_c, i_{c'}), p(o_{c'}, o_c), p(i_c, i_{c''}), p(o_{c''}, o_c). \qquad (2)\end{aligned}$$

for each position i on the tape, and similarly when the head is moving to the right.

When $g(q) = \wedge$, we associate with each transition $\delta(q,\gamma) = \{(q',\gamma',L),(q'',\gamma'',L)\}$ the following rule:

$$\begin{aligned}&\mathit{conf}(i_c, q, cell_1, cur_1, \ldots, cell_i, 0, \gamma, 1, \ldots, o_c) \to \\ &\quad \exists i_{c'}, o_{c'}, i_{c''}, o_{c''}\ \mathit{conf}(i_{c'}, q', cell_1, cur_1, \ldots, cell_i, 1, \gamma', 0, \ldots, o_{c'}), \\ &\qquad \mathit{conf}(i_{c''}, q'', cell_1, cur_1, \ldots, cell_i, 1, \gamma'', 0, \ldots, o_{c''}), \\ &\qquad\qquad p(i_c, i_{c'}), p(o_{c'}, i_{c''}), p(o_{c''}, o_c). \qquad (3)\end{aligned}$$

Figure 1 illustrates the functioning of rules of types (2) and (3). We denote by $\mathcal{R}_{\mathcal{M},x}$ the set containing all the rules defined above [4]. The above rules (where input and output positions are removed) simulate the run of a PSPACE TM [12].

The following property formalizes the reduction and establishes its correctness.

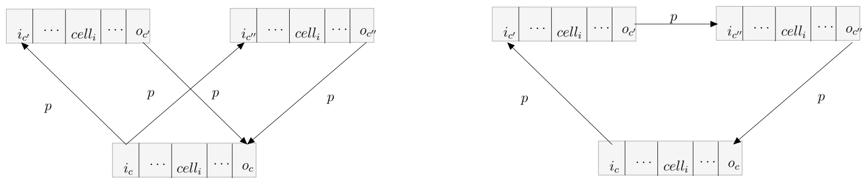

Fig. 1. Existential (left) and universal (right) gadgets

Property 7. Let $\mathcal{M}$ be an alternating PSPACE Turing machine, and let α be an atom of $\mathit{chase}(I_{\mathcal{M},x},\mathcal{R}_{\mathcal{M},x})$ representing a configuration $c(\alpha)$. Then $c(\alpha)$ is an accepting configuration of $\mathcal{M}$ if and only if there is a path in $\mathit{chase}(I_{\mathcal{M},x},\mathcal{R}_{\mathcal{M},x})$ from $i(\alpha)$ to $o(\alpha)$ whose label belongs to p^*.

Proof. ($\Leftarrow$) Let $\alpha \in \mathit{chase}(I_{\mathcal{M},x},\mathcal{R}_{\mathcal{M},x})$ represent a configuration $c(\alpha)$, and let C_α be the restriction of $\mathit{chase}(I_{\mathcal{M},x},\mathcal{R}_{\mathcal{M},x})$ to α. We show by induction on the number of atoms of C_α that the required path exists. Note that the induction is well-founded as the Skolem chase is finite (recall that the considered Turing machines terminate).

[4] Note that x is required to determine the arity of *conf*.

– If C_α contains one atom, then there can be no path in $\mathit{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$ witnessing $p^*(i(\alpha), o(\alpha))$. Suppose then that C_α contains two atoms. In this case, the only atom in C_α other than α must be $p(i(\alpha), o(\alpha))$. The only way to derive such an atom is to apply a rule of the form (1), which is applied if and only if $c(\alpha)$ is in an accepting state, hence $c(\alpha)$ is an accepting configuration of $\mathcal{M}$.
– Next assume that the result holds for any atom α such that C_α has less than n atoms, and let α be an atom such that C_α contains n atoms. We distinguish two cases:
  - Case 1: the state of $c(\alpha)$ is existential. Then, since the rules of type (2) must be satisfied, C_α contains atoms α_1 and α_2 representing the successor configurations of $c(\alpha)$. The existence of a path from $i(\alpha)$ to $o(\alpha)$ implies that there is either a path from $i(\alpha_1)$ to $o(\alpha_1)$ or a path from $i(\alpha_2)$ to $o(\alpha_2)$. To see why, observe that every p-atom involving $i(\alpha)$ or $o(\alpha)$ is added either by the same rule application as created α or by a rule of type (2) applied to α. Only atoms of the second kind (refer to Fig. 1, left) can belong to a shortest path from $i(\alpha)$ to $o(\alpha)$, as atoms of the first kind have $i(\alpha)$ (resp. $o(\alpha)$) as second (resp. first) argument. If we have a path from $i(\alpha_1)$ to $o(\alpha_1)$, then we can apply the induction assumption to α_1 to get that $c(\alpha_1)$ is an accepting configuration, which implies that $c(\alpha)$ is also accepting. We can proceed analogously if we have path from $i(\alpha_2)$ to $o(\alpha_2)$.
  - Case 2: the state of $c(\alpha)$ is universal. As the rules of type (3) must be satisfied, the existence of a path from $i(\alpha)$ to $o(\alpha)$ implies the existence of a path from $i(\alpha_1)$ to $o(\alpha_1)$ and a path from $i(\alpha_2)$ to $o(\alpha_2)$, where α_1 and α_2 represent the successor configurations of $c(\alpha)$ (refer to Fig. 1, right). By the induction assumption, $c(\alpha_1)$ and $c(\alpha_2)$ are both accepting configurations, which means that $c(\alpha)$ is also accepting.

($\Rightarrow$) We prove the other direction by induction on the number of transitions that need to be performed to prove that $c(\alpha)$ is accepted by $\mathcal{M}$.

– If no transitions are required, this means that $c(\alpha)$ is in an accepting state. Thus, Rule (1) is applicable, and $p(i(\alpha), o(\alpha))$ is present in $\mathit{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$.
– Assume the result holds up to n required transitions. We distinguish two cases:
  - Case 1: the state of $c(\alpha)$ is existential. As $c(\alpha)$ is accepting, this means that one of its two successor configurations, say $c(\alpha_1)$, is accepting. Moreover, the number of transitions required to accept $c(\alpha_1)$ is strictly smaller than for $c(\alpha)$. By the induction assumption, $p^*(i(\alpha_1), o(\alpha_1))$ is present in $\mathit{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. As $p(i(\alpha), i(\alpha_1))$ and $p(o(\alpha_1), o(\alpha))$ are also present (since the rules of the form (2) generate them), this proves that $p^*(i(\alpha), o(\alpha))$ is present as well.
  - Case 2: the state of $c(\alpha)$ is universal. As $c(\alpha)$ is accepting, this means that its two successor configuration are also accepting. By the induction assumption, this means that $p^*(i(\alpha_1), o(\alpha_1))$ and $p^*(i(\alpha_2), o(\alpha_2))$ are present in $\mathit{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. As the rules of the form (3) also generate $p(i(\alpha), i(\alpha_1))$, $p(o(\alpha_1), i(\alpha_2))$, and $p(o(\alpha_2), o(\alpha))$, this proves that $p^*(i(\alpha), o(\alpha))$ is present in $\mathit{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. □

Now let $\mathcal{M}$ be an alternating PSPACE Turing machine, x be an input to $\mathcal{M}$, and α be the unique atom in $I_{\mathcal{M},x}$. Then by Property 7, $c(\alpha)$ is an accepting configuration of $\mathcal{M}$ if and only if $I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x} \models p^*(i(\alpha), o(\alpha))$. This, together with known results, yields the following lower bounds:

Theorem 2. RPQ *Answering in the presence of linear existential rules is* NL-*hard in data complexity,* PTIME-*hard in combined complexity with bounded arity and* EXPTIME-*hard in combined complexity without arity bound, even for a fixed RPQ.*

Note that the preceding reduction can be easily adapted to show that atomic query answering under rulesets containing linear rules and transitivity rules is EXPTIME-hard. Assuming EXPTIME≠PSPACE, this result is in contradiction with Theorem 5 in [2], which purports to show a PSPACE upper bound. Indeed, after reexamining the proofs, the authors of the latter work have identified the flaw, which occurs in the analysis of the combined complexity of their rewriting-based decision procedure. It turns out that the procedure runs in exponential time, rather than in polynomial space (the NL upper bound in data complexity remains valid). Combining our lower bound with their procedure shows that the problem is EXPTIME-complete in combined complexity.

5 Conclusion and Future Work

In this paper, we have investigated the complexity of evaluating regular path queries under linear existential rules. We have shown that it is NL-complete in data complexity, PTIME-complete in combined complexity when the predicate arity is bounded, and EXPTIME-complete otherwise. This behavior is somewhat surprising with respect to prior work: indeed, for DL-Lite$_{\mathcal{R}}$, the combined complexity of RPQ answering is lower than for CQs, whereas we observe just the opposite in the linear case (recall CQ answering is PSPACE-complete under linear rules). The upper bound was shown by adapting an existing decision procedure for DL-Lite, using a refined definition of type. The lower bound builds upon a PSPACE-hardness result for CQ answering under linear rules.

There are two natural ways to extend the present work: either investigate more expressive forms of path queries (with conjunction and/or nesting) over linear rules, or consider the effect of moving to more expressive decidable classes of existential rules.

Acknowledgements. This work was supported by the ANR project 12 JS02 007 01.

References

1. Baget, J., Leclère, M., Mugnier, M., Salvat, E.: Extending decidable cases for rules with existential variables. In: Proc. of IJCAI. pp. 677–682 (2009)
2. Baget, J., Bienvenu, M., Mugnier, M., Rocher, S.: Combining existential rules and transitivity: Next steps. In: Proc. of IJCAI. pp. 2720–2726 (2015)
3. Bienvenu, M., Calvanese, D., Ortiz, M., M.Šimkus: Nested regular path queries in description logics. In: Proc. of KR (2014)
4. Bienvenu, M., Ortiz, M.: Ontology-mediated query answering with data-tractable description logics. In: Reasoning Web. pp. 218–307 (2015)
5. Bienvenu, M., Ortiz, M., Simkus, M.: Regular path queries in lightweight description logics: Complexity and algorithms. J. Artif. Intell. Res. (JAIR) 53, 315–374 (2015)

6. Bourhis, P., Krötzsch, M., Rudolph, S.: How to best nest regular path queries. In: Proc. of DL. pp. 404–415 (2014)
7. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: Proc. of KR. pp. 70–80 (2008)
8. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics: An automata-theoretic approach. In: Proc. of AAAI. pp. 391–396 (2007)
9. Calvanese, D., Eiter, T., Ortiz, M.: Regular path queries in expressive description logics with nominals. In: Proc. of IJCAI. pp. 714–720 (2009)
10. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics via alternating tree-automata. Inf. Comput. 237, 12–55 (2014)
11. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: Proc. of PODS (1998)
12. Gottlob, G., Papadimitriou, C.H.: On the complexity of single-rule datalog queries. Inf. Comput. 183(1), 104–122 (2003)
13. Grau, B.C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity notions for existential rules and their application to query answering in ontologies. J. Artif. Intell. Res. (JAIR) 47, 741–808 (2013)
14. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. 28(1), 167–189 (1984)
15. Kostylev, E.V., Reutter, J.L., Vrgoc, D.: XPath for DL ontologies. In: Proc. of AAAI (2015)
16. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. 4(4), 455–469 (1979)
17. Marnette, B.: Generalized schema-mappings: from termination to tractability. In: Proc. of PODS. pp. 13–22 (2009)
18. Ortiz, M., Rudolph, S., Šimkus, M.: Query answering in the Horn fragments of the description logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. In: Proc. of IJCAI (2011)
19. Stefanoni, G., Motik, B., Krötzsch, M., Rudolph, S.: The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. J. of Art. Intell. Res. (JAIR) 51, 645–705 (2014)