Answering Path Queries under Linear and Guarded Existential Rules

JEAN-FRANÇOIS BAGET, LIRMM, Inria, University of Montpellier, CNRS, France
MEGHYN BIENVENU, Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France
MARIE-LAURE MUGNIER, LIRMM, Inria, University of Montpellier, CNRS, France
MICHAËL THOMAZO, Inria, DIENS, ENS, PSL University, CNRS, France

Ontology-mediated query answering is concerned with the problem of answering queries over knowledge bases consisting of a database instance and an ontology. While most work in the area focuses on conjunctive queries (CQs), navigational queries have gained increasing attention. In this paper, we investigate the complexity of answering two-way (conjunctive) regular path queries ((C)RPQs) over knowledge bases whose ontology is given by a set of guarded existential rules. We first consider the subclass of linear existential rules and show that (C)RPQ answering is NL-complete in data complexity, which matches the data complexity of answering RPQs over plain graph databases (i.e., without an ontology). In combined complexity, both tasks are ExpTime-complete in the general case, but RPQ and CRPQ answering drop to PTime-complete and PSpace-complete respectively if there is a bound on predicate arity. For guarded rules, we provide a non-trivial reduction to the linear case, which allows us to show that the complexity of (C)RPQ answering is the same as for CQs, namely 2ExpTime-complete in combined complexity (ExpTime-complete in the bounded-arity case) and PTime-complete in data complexity.

JAIR Associate Editor: Roberta Calegari

JAIR Reference Format:
Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Michaël Thomazo. 2026. Answering Path Queries under Linear and Guarded Existential Rules. *Journal of Artificial Intelligence Research* 86, Article 41 (July 2026), 54 pages. DOI: 10.1613/jair.1.19922

1 Introduction

Over the past two decades, significant research efforts have been devoted to *ontology-mediated query answering* (OMQA), in which data is enriched with an ontology that expresses domain knowledge, and queries are answered by taking both the data and the ontology into account (see e.g. (Poggi et al. 2008) for a seminal early work and (Xiao et al. 2018) for a short survey). Adding an ontological layer on top of data allows a user to formulate queries in a more familiar way, closer to their conceptualization of the application domain, thereby abstracting from the actual structure of databases, which may have been designed for independent purposes. On the other hand, it can also provide more complete answers to queries, as answers are based not only on facts explicitly stored in the data but also on those that are consequences of the ontology and the data. Finally, in the context of data integration, it provides a uniform mediating layer that facilitates the integration of heterogeneous data sources. The OMQA approach thus offers several important advantages, which are of relevance in diverse application areas (we refer readers to (Xiao et al. 2019) for an overview of OMQA applications). However, the need to take ontological

Authors' Contact Information: Jean-François Baget, ORCID: 0000-0001-7221-5770, baget@lirmm.fr, LIRMM, Inria, University of Montpellier, CNRS, Montpellier, France; Meghyn Bienvenu, ORCID: 0000-0001-6229-8103, meghyn.bienvenu@labri.fr, Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, Talence, France; Marie-Laure Mugnier, ORCID: 0000-0002-0574-3693, mugnier@lirmm.fr, LIRMM, Inria, University of Montpellier, CNRS, Montpellier, France; Michaël Thomazo, ORCID: 0000-0002-1437-6389, michael.thomazo@inria.fr, Inria, DIENS, ENS, PSL University, CNRS, Paris, France.

DOI: 10.1613/jair.1.19922

information into account when computing query answers can make the query answering task significantly more complex, depending on the expressivity of the ontology and query languages.

Two families of ontology languages are mainly considered in OMQA, namely description logics (DLs) and existential rules (see e.g. the survey chapters (Ortiz and Simkus 2012, Kontchakov et al. 2013, Bienvenu and Ortiz 2015) on OMQA with DLs and (Calì et al. 2009a, Mugnier and Thomazo 2014) for existential rules). These ontology languages allow one to reason in open domains, which means that existing entities are not supposed to be restricted to those explicitly encoded in the data. Existential rules, also known as tuple generating dependencies (Abiteboul et al. 1994), can be seen as an extension of Datalog (equivalently, function-free Horn rules) with existentially quantified variables in the rule heads. For instance, the following existential rule expresses that 'every person has a parent who is a person': $\forall x.\ human(x) \rightarrow \exists y.\ hasParent(x, y) \wedge human(y)$. Applied to a fact like $human(a)$, this rule leads to infer the existence of an infinite number of entites. Existential rules generalize most DLs that are used in the context of data access, often referred to as Horn description logics. More generally, they overcome some well-known limitations of DLs by their unrestricted predicate arity and their ability to express non-tree-shaped relationships between atoms. This added expressivity makes ground fact entailment undecidable (e.g. (Beeri and Vardi 1981)), however many decidable classes of existential rules have been exhibited, which offer different tradeoffs between expressivity and reasoning complexity. We focus here on two central classes (Calì et al. 2009b, 2013): *guarded* existential rules, in which the body of a rule has a guard, i.e., an atom that contains all the variables occurring in the rule body, and *linear* existential rules, a subclass of guarded rules in which the body of a rule is made of a single atom, as in the above example rule.

As for the query language, most work on OMQA has focused on *conjunctive queries* (CQs), a fundamental query class in relational databases and in the Semantic Web, where this class is also known as basic graph pattern queries. However, the increasing popularity of graph databases and the rise of knowledge graphs have shifted attention towards navigational queries (Angles et al. 2017, Libkin et al. 2025), which are able to capture structural patterns of unbounded size. The simplest such queries are *regular path queries* (RPQs) (Cruz et al. 1987, Mendelzon and Wood 1995, Calvanese et al. 2002), which ask for paths of potentially arbitrary length whose label conforms to a given regular language, hence allowing for a controlled form of recursion over binary predicates. Many extensions of RPQs have been investigated, including *conjunctive regular path queries* (CRPQs) (Florescu et al. 1998, Calvanese et al. 2000, Cucumides et al. 2023) which generalize both RPQs and CQs. In particular, the core queries of SPARQL 1.1 (the W3C standard[1] for querying RDF data) roughly correspond to CRPQs. The next example illustrates these different types of queries in the context of ontology-mediated query answering with existential rules. It will be used as a running example throughout the paper to provide a high-level overview of the key notions. Additional dedicated examples will illustrate the details and subtilities underlying the technical constructions.

Example 1 (Running Example). *Consider data describing relationships in a social network, which uses the predicates* follows*(x,y),* isFriendOf*(x,y) and* message*(m, x, y), meaning that message m was sent from x to y. Asking for all pairs (x, y) such that x and y both follow some person can be expressed as a CQ:*

$$q_1(x, y) = \exists z.\ \mathsf{follows}(x, z) \wedge \mathsf{follows}(y, z)$$

It can also be expressed as an RPQ:

$$q_2(x, y) = \mathsf{follows} \cdot \mathsf{follows}^{-}(x, y)$$

where the operator $^{-}$ denotes the inverse relation. Asking for pairs (x, y) such that y is a direct or indirect follower of x can be expressed by an RPQ but not by a CQ:

$$q_3(x, y) = \mathsf{follows}^{-} \cdot (\mathsf{follows}^{-})^{*}(x, y)$$

[1] SPARQL 1.1 Query Language W3C Recommendation: http://www.w3.org/TR/sparql11-query

In turn, asking for pairs (x, y) *such that* x *and* y *follow each other can be expressed by a CQ but not by an RPQ:*

$$q_4(x, y) = \mathsf{follows}(x, y) \wedge \mathsf{follows}(y, x)$$

Now, to retrieve pairs (x, y) *such that* x *follows* y *and* y *directly or indirectly follows* x *requires the expressivity of a CRPQ, which generalizes* q_4 *by replacing a standard predicate with a path expression:*

$$q_5(x, y) = \mathsf{follows}(x, y) \wedge \mathsf{follows} \cdot \mathsf{follows}^*(y, x)$$

Asking whether user Alice *received a message from someone she follows directly or indirectly can be expressed by a (Boolean) CRPQ:*

$$q_6() = \exists y, m.\ \mathsf{follows} \cdot \mathsf{follows}^*(\mathit{Alice}, y) \wedge \mathit{message}(m, y, \mathit{Alice})$$

Now, let us add ontological knowledge stating that (1) the relation isFriendOf *is symmetric, (2) it is a specific case of the relation* follows*, and (3) following someone implies sending her a message. This can be expressed using the following three linear rules:*

(ρ_1) $\forall x, y.\ \mathsf{isFriendOf}(x, y) \rightarrow \mathsf{isFriendOf}(y, x)$
(ρ_2) $\forall x, y.\ \mathsf{isFriendOf}(x, y) \rightarrow \mathsf{follows}(x, y)$
(ρ_3) $\forall x, y.\ \mathsf{follows}(x, y) \rightarrow \exists m.\ \mathsf{message}(m, x, y)$

Assume the data indicates that Bob *follows* Alice *and they have a common friend* Carmen*, which is expressed by the facts* $\mathsf{follows}(\mathsf{Bob}, \mathsf{Alice})$, $\mathsf{isFriendOf}(\mathsf{Carmen}, \mathsf{Alice})$ *and* $\mathsf{isFriendOf}(\mathsf{Carmen}, \mathsf{Bob})$*. None of the preceding queries have an answer on this data alone, while all of them have answers when the rules are taken into account. Indeed, the following facts are inferred from the data and the rules:*

$\mathsf{isFriendOf}(\mathsf{Alice}, \mathsf{Carmen})$, $\mathsf{isFriendOf}(\mathsf{Bob}, \mathsf{Carmen})$
$\mathsf{follows}(\mathsf{Carmen}, \mathsf{Alice})$, $\mathsf{follows}(\mathsf{Carmen}, \mathsf{Bob})$, $\mathsf{follows}(\mathsf{Alice}, \mathsf{Carmen})$, $\mathsf{follows}(\mathsf{Bob}, \mathsf{Carmen})$
$\mathsf{message}(\mathsf{m}_0, \mathsf{Bob}, \mathsf{Alice})$, $\mathsf{message}(\mathsf{m}_1, \mathsf{Carmen}, \mathsf{Alice})$, $\mathsf{message}(\mathsf{m}_2, \mathsf{Carmen}, \mathsf{Bob})$,
$\mathsf{message}(\mathsf{m}_3, \mathsf{Alice}, \mathsf{Carmen})$, $\mathsf{message}(\mathsf{m}_4, \mathsf{Bob}, \mathsf{Carmen})$

For instance, the facts $\mathsf{follows}(\mathsf{Alice}, \mathsf{Carmen})$ *and* $\mathsf{follows}(\mathsf{Carmen}, \mathsf{Bob})$ *yield the answer* $(\mathsf{Bob}, \mathsf{Alice})$ *to* q_3*; from these two facts and* $\mathsf{message}(\mathsf{m}_0, \mathsf{Bob}, \mathsf{Alice})$*, we conclude that* q_6 *is answered positively.*

As an example of a guarded rule, consider the following rule defining the predicate isPaired*:*

(ρ_4) $\forall x, y.\ \mathsf{follows}(x, y) \wedge \mathsf{follows}(y, x) \rightarrow \mathsf{isPaired}(x, y)$

Finally, the rules below define the predicate extFollows*, which captures the regular expression* $\mathsf{follows}.\mathsf{follows}^*$*:*

$\forall x, y.\ \mathsf{follows}(x, y) \rightarrow \mathsf{extFollows}(x, y)$
$\forall x, y, z.\ \mathsf{follows}(x, y) \wedge \mathsf{extFollows}(y, z) \rightarrow \mathsf{extFollows}(x, z)$

Using the predicate extFollows*, the above (C)RPQs can be reformulated as CQs. Note however that the last rule is not guarded, hence does not belong to the existential rule fragments studied in this paper. This shows that navigational features in queries can compensate for certain limitations of the selected ontological fragment.*

As description logics are well suited to expressing ontological knowledge about graph-like data, such as ABoxes and RDF graphs, it is not surprising that navigational queries have long been investigated for DL ontologies. Such studies cover a variety of DLs, ranging from highly expressive DLs of the $\mathcal{Z}$ family (Calvanese et al. 2007b, 2009, 2014), to Horn DLs like Horn-$\mathcal{SROIQ}$ (Ortiz et al. 2011, Bienvenu et al. 2014) and lightweight DLs of the DL-Lite and $\mathcal{EL}$ families (Stefanoni et al. 2014, Kostylev et al. 2015, Bienvenu et al. 2015). While the main focus of the preceding work was on establishing the complexity of answering navigational queries, there has also been some very recent work exploring how to obtain practical algorithms (Dragovic et al. 2023, Löhnert et al. 2025).

By allowing predicates of any arity, in addition to the expression of cyclic relations between entities, existential rules naturally apply not only to graph data but also to data with a more complex hypergraph structure, such as relational databases or graph data enriched with contextual information. This flexibility is particularly useful in

Table 1. Landscape of (C)(RP)Q Answering Complexity under Linear Rules

	RPQ	**CQ**	**CRPQ**
Data	NL-c	AC_0	NL-c
Combined, bounded arity	PTime-c	NP-c	PSpace-c
Combined, unbounded arity	ExpTime-c	PSpace-c	ExpTime-c

Table 2. Landscape of (C)(RP)Q Answering Complexity under Guarded Rules

	RPQ	**CQ**	**CRPQ**
Data	PTime-c	PTime-c	PTime-c
Combined, bounded arity	ExpTime-c	ExpTime-c	ExpTime-c
Combined, unbounded arity	2ExpTime-c	2ExpTime-c	2ExpTime-c

applications involving heterogeneous data, since it makes it possible to handle each kind of data as it is, without breaking it down into binary facts. This does not contradict the observation that binary relationships are still central in any context, whether they are found directly in certain data or built as part of the ontological modeling. Hence, navigational queries remain highly relevant even when the data and/or ontology includes higher-arity relations. However, while there is now an extensive literature on navigational queries in the presence of DL ontologies, very little is known about the complexity (or even decidability) of answering such queries under different kinds of existential rules.

Our paper makes an important contribution towards clarifying the complexity landscape for navigational queries under existential rules by establishing the complexity of answering (C)RPQs in the presence of linear and guarded existential rulesets. To the best of our knowledge, these results (which were first reported in the conference papers (Bienvenu and Thomazo 2016, Baget et al. 2017)) constitute the first and the only complexity results targeting navigational queries with existential rules. However, two recent works have established the decidability of (i) RPQ answering for sticky existential rulesets (Ostropolski-Nalewaja and Rudolph 2024), and (ii) answering an extension of CRPQs for finite clique-width rulesets (Feller et al. 2023) (in both cases without any upper complexity bounds), demonstrating the continued interest in the topic.

Contributions. We obtain tight complexity results for both combined and data complexities over guarded rules and their linear subclass for both RPQ and CRPQ answering. With respect to combined complexity, we furthermore distinguish between bounded and unbounded predicate arity. These results are synthesized in Table 1 for linear rules and in Table 2 for guarded rules, alongside existing results for CQs (the grey background indicates new results). In the linear case, we see that CRPQ answering and RPQ answering have the same data and unbounded-arity combined complexities, and these complexities are higher than those of CQ answering. However, for bounded-arity combined complexity, RPQ answering is easier than CQ answering, while CRPQ is more difficult. In the guarded case, we show that RPQ answering and CRPQ answering both have the same complexity as CQ answering, regardless of the complexity measure. All these results are proven for existential rules with a single head atom; since arbitrary linear rules (respectively guarded rules) can be polynomially translated into atomic-head linear rules (respectively guarded rules), the results also hold for combined complexity with unbounded predicate arity as well as data complexity. Even though the translation does not preserve bounded predicate arity, we show that for arbitrary linear rules the same upper bounds apply (and we conjecture this is true also for guarded rules). It is also worth noting that, while our complexity results for (C)RPQs are in most cases higher than the analogous results for plain graph databases, the NL data complexity of (C)RPQ answering with linear rules coincides with the data complexity of (C)RPQs in the ontology-free setting.

Paper organization. After a preliminary section, we first investigate the complexity of query answering for RPQs and linear existential rules. We rely on a forward chaining scheme, known as the *chase* (Maier et al. 1979, Beeri and Vardi 1984) which, starting from a (finite) set of facts, iteratively applies rules until a fixpoint is reached, if any. To get complexity upper bounds, we provide an algorithm that exploits the structure of paths of terms in the chase to guess a path that complies with the language defined by the RPQ (Section 3). We then prove that this algorithm is worst-case optimal: in the bounded arity case by using known lower bounds for DL-Lite, a language in which relevant assertions are specific linear rules (Bienvenu et al. 2015), and in the general case by providing a novel reduction from an alternating PSpace Turing machine to RPQ answering under linear rules (Section 4). We then consider CRPQ answering, still under linear rules (Section 5). We devise an algorithm that exploits additional structural properties of the chase and uses the previous RPQ answering algorithm as an oracle, which allows us to upper-bound the problem complexity. As the obtained upper bounds match lower bounds coming from previous results, this algorithm is worst-case optimal. Finally, to study the complexity of CRPQ answering under guarded rules, we provide a non-trivial reduction of the guarded case to the linear case (Section 6). This translation involves a double exponential blow-up of the set of rules (while the instance only grows exponentially in the predicate arity). However, a careful analysis of the algorithm provided for CRPQ answering under linear rules shows that it actually runs in 2ExpTime with respect to the input guarded knowledge base (and in ExpTime in the case of bounded-arity rules). We end with a presentation of closely related work (Section 7) and remaining questions (Section 8).

This article is an extended version of two previously published conference papers (Bienvenu and Thomazo 2016, Baget et al. 2017). With respect to these conference papers, we provide a unified presentation, full proofs of the results, detailed examples and a review of related work with the latest results.

2 Preliminaries

We consider logical vocabularies of the form $\mathcal{V} = (\mathcal{P}, C)$, where $\mathcal{P}$ is a finite set of predicates and C is an infinite set of constants. A (standard) *atom* α has the form $r(\mathbf{t})$ where r is a predicate of arity n and $\mathbf{t}$ is a tuple of terms (i.e., variables or constants) with $|\mathbf{t}| = n$. For $1 \leq i \leq |\mathbf{t}|$, we denote by $\alpha[i]$ the term at position i in α. We denote by $\mathsf{terms}(\alpha)$ (resp. $\mathsf{vars}(\alpha)$) the set of terms (resp. variables) in α and extend the notations to a set of atoms. A *ground* atom contains only constants. A formula is *atomic* if it is a single atom. It is *closed* if it has no free variable. Given a (possibly infinite) set of atoms A and a (possibly infinite) set of terms T, $A_{|T}$ denotes the restriction of A to atoms α with $\mathsf{terms}(\alpha) \subseteq T$.

An *interpretation* of a vocabulary $\mathcal{V} = (\mathcal{P}, C)$ is denoted by $\mathcal{I} = (\Delta_{\mathcal{I}}, .^{\mathcal{I}})$, where $\Delta_{\mathcal{I}}$ is the possibly infinite (non-empty) domain of $\mathcal{I}$ and $.^{\mathcal{I}}$ is the interpretation function, such that $a^{\mathcal{I}} \in \Delta$ for each $a \in C$ and $r^{\mathcal{I}} \subseteq \Delta^n$ for each predicate $r \in \mathcal{P}$ with arity n. An interpretation is a *model* of a closed formula F (resp. a set of closed formulas F) if it makes F (resp. every formula in F) true. For F a conjunction of atoms, a *match* of F in $\mathcal{I}$ is a mapping π from $\mathsf{terms}(F)$ to elements of $\Delta_{\mathcal{I}}$ such that *(i)* $\pi(a) = a^{\mathcal{I}}$ for each constant a; *(ii)* $\pi(\mathbf{t}) \in r^{\mathcal{I}}$ for each atom $r(\mathbf{t})$ in F. Then, it holds that $\mathcal{I}$ is a model of a closed formula F iff there is a *match* of F in $\mathcal{I}$. We consider classical logical entailment: given a set of closed formulas F and a closed formula f, $F \models f$ means that every model of F is a model of f. In the following, we identify (the existential closure of) a conjunction of atoms with the set of these atoms. Given two sets of atoms A_1 and A_2, a *homomorphism* from A_2 to A_1 is a substitution π of $\mathsf{vars}(A_2)$ by $\mathsf{terms}(A_1)$ such that $\pi(A_2) \subseteq A_1$. An *isomorphism* from A_2 to A_1 is a bijective substitution b of $\mathsf{vars}(A_2)$ by $\mathsf{vars}(A_1)$ such that $b(A_2) = A_1$.

Given two existentially closed conjunctions of atoms A_1 and A_2, it holds that $A_1 \models A_2$ iff there is a homomorphism from (the set of atoms in) A_2 to (the set of atoms in) A_1. Hence, A_1 and A_2 are logically equivalent if and only if they are homomorphically equivalent. It is worth noticing that equivalent sets of atoms are not necessarily isomorphic.

2.1 Existential Rule Knowledge Bases

An *instance* is a finite set of ground atoms. An *extended instance* is a finite set of arbitrary atoms, logically translated into an existentially-closed conjunction of atoms. An *existential rule* ρ (or simply *rule*) is of the form $\forall \mathbf{x} \forall \mathbf{y}\ [\ B(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ H(\mathbf{x}, \mathbf{z})\ ]$, where B and H are non-empty conjunctions of atoms on variables, respectively called the *body* and the *head* of ρ, and $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are pairwise disjoint. We also denote the body and the head of a rule ρ by $\mathsf{body}(\rho)$ and $\mathsf{head}(\rho)$, respectively. We make the common assumption that rules do not contain constants, which simplifies technical tools. The variables of $\mathbf{x}$ (resp. $\mathbf{z}$) are called *frontier variables* (resp. *existential variables*). For brevity, we denote by $B \rightarrow H$ a rule with body B and head H and in our examples universal quantifiers are implicit.

A *knowledge base* (KB) is of the form $\mathcal{K} = (I, \mathcal{R})$, where I is an instance and $\mathcal{R}$ a set of existential rules. In the following, we assume that distinct rules in $\mathcal{R}$ have disjoint sets of variables, even if we reuse variables in examples for the sake of simplicity.

A rule $\rho = B \rightarrow H$ is *applicable* to a set of atoms A if there is a homomorphism π from B to A. The pair (ρ, π) is called a *trigger* on A. The application of ρ according to π (or: the application of the trigger (ρ, π)) produces a set of atoms obtained from $\mathsf{head}(\rho)$ by replacing each frontier variable x with $\pi(x)$ and each existential variable with a fresh variable, usually called a *null*. We denote by π^{safe} this extension of π that "safely" renames existential variables, so that distinct applications of the same rule produce disjoint sets of nulls. The resulting set of atoms is $A \cup \pi^{\text{safe}}(H)$. An atom $\alpha \in \pi^{\text{safe}}(H)$ is said to be *(directly) generated* by the trigger (ρ, π) if $\alpha \notin A$. Observe that an atom α that is produced by a trigger (ρ, π) without being "generated" by that trigger necessarily comes from an atom in $\mathsf{head}(\rho)$ that does not contain any existential variable.

The fundamental tool for reasoning on existential rules is a forward chaining procedure known as the *chase*. Briefly, the chase enriches a given instance by applying rules until a fixpoint is reached. This process may be infinite, as for instance with $\mathcal{R} = \{h(x) \rightarrow \exists z\ p(x, z) \wedge h(z)\}$ ("every human has a parent who is a human") and $I = \{h(a)\}$. Several variants of the chase are known (see e.g., (Grahne and Onet 2018)). We consider here the simplest variant, called the *oblivious chase* (Calì et al. 2013).

Formally, an $\mathcal{R}$-*derivation* from an (extended) instance I is a possibly infinite sequence of (extended) instances and triggers $D = I_0 (= I)\ (\rho_1, \pi_1)\ I_1 \ldots\ (\rho_n, \pi_n)\ I_n, \ldots$, where, for all $i \geq 1$, I_i results from the application of the trigger (ρ_i, π_i) on I_{i-1}, with $\rho_i \in \mathcal{R}$, and no trigger appears twice in D. It is simply called a derivation when $\mathcal{R}$ is clear from the context; and it is written $I_0 (= I)\ I_1 \ldots\ I_n, \ldots$ when the triggers are not needed. The *result* of D is the set of atoms obtained along the sequence, i.e., $\bigcup_{i \geq 0} I_i$, and we denote it by $\mathit{atoms}(D)$. The derivation D is *fair* if, for any I_i and trigger (ρ, π) on I_i, there is I_j in D such that I_j results from the application of (ρ, π). An (oblivious) *chase sequence* of $\mathcal{K} = (I, \mathcal{R})$ is a fair $\mathcal{R}$-derivation from I.

It is known that oblivious chase sequences on a given KB are either all finite or all infinite, and that they result in isomorphic sets of atoms (Calì et al. 2013). Hence, given a KB $\mathcal{K} = (I, \mathcal{R})$, we denote by $\mathsf{chase}(\mathcal{K})$, or $\mathsf{chase}(I, \mathcal{R})$, the result of any oblivious chase sequence of $\mathcal{K}$. Furthermore, the name chase is classically used to denote both the forward chaining process and its result.

The (result of the) chase of $\mathcal{K}$ can be seen as a logical interpretation, which is a model of $\mathcal{K}$. This model $\mathcal{I}_{\mathcal{K}} = (\Delta_{\mathcal{I}_{\mathcal{K}}}, .^{\mathcal{I}_{\mathcal{K}}})$ has for domain $\Delta_{\mathcal{I}_{\mathcal{K}}} = \mathsf{terms}(\mathsf{chase}(\mathcal{K}))$ and its interpretation function $.^{\mathcal{I}_{\mathcal{K}}}$ is defined by the atoms in $\mathsf{chase}(\mathcal{K})$ (i.e., for each constant c, $c^{\mathcal{I}_{\mathcal{K}}} = c$ holds, and, for each predicate p, $p^{\mathcal{I}_{\mathcal{K}}}$ is the set of tuples $(t_1, \ldots t_k)$ such that $p(t_1, \ldots t_k) \in \mathsf{chase}(\mathcal{K})$). Furthermore, $\mathcal{I}_{\mathcal{K}}$ has the fundamental property of being a *universal* model of $\mathcal{K}$, i.e., it homomorphically maps to any other model of $\mathcal{K}$ (Fagin et al. 2005, Deutsch et al. 2008). It follows that it can act as a representative of all models of $\mathcal{K}$ to check entailment of conjunctive queries, and the more general conjunctive regular path queries, as explained next (see Section 2.3).

We will consider two specific classes of existential rules, namely linear and guarded. A rule is *linear* if its body is atomic. A set of atoms A is *guarded* if it contains an atom α, called a guard, such that $\mathsf{terms}(\alpha) = \mathsf{terms}(A)$.

Since A may contain several guards, we use the notation (A, α) to specify that α is the considered guard. A rule is *guarded* if its body is guarded. Note that a linear rule is trivially guarded.

Single-head translation. In the following, we will make the assumption that the head of an existential rule is atomic. It is well-known that any existential rule ρ on a vocabulary $\mathcal{V}$ can be decomposed into a set of atomic-head rules by adding a fresh predicate p_ρ, whose arity is the number of variables in the head of ρ. E.g., the rule $\rho = r(x, y) \rightarrow \exists z\, q(x, z) \wedge q(y, z)$ can be decomposed into three atomic-head rules: $r(x, y) \rightarrow \exists z\, p_\rho(x, y, z)$; $p_\rho(x, y, z) \rightarrow q(x, z)$; $p_\rho(x, y, z) \rightarrow q(y, z)$. This polynomial translation preserves entailment of formulas on $\mathcal{V}$. Moreover, the decomposition of a guarded (resp. linear) rule yields a set of guarded (resp. linear) rules. Hence, the assumption that rules have an atomic head can be made without loss of generality regarding combined complexity with unbounded predicate arity, as well as data complexity.

2.2 Properties of Linear Rules

A well-known property of linear rules is that chasing independently each atom of an instance yields a result equivalent to the chase of this instance, i.e., $\cup_{\alpha \in I} \mathsf{chase}(\{\alpha\}, \mathcal{R})$ and $\mathsf{chase}(I, \mathcal{R})$ are homomorphically equivalent. Indeed, the key property of linear rules is that a pair (ρ, π) is a trigger on a set of atoms A iff there is $\alpha \in A$ such that (ρ, π) is a trigger on $\{\alpha\}$. Hence, the sequence of triggers in a derivation from I can be split into sequences of triggers that each define a derivation from an atom in I. It follows that $\mathsf{chase}(I, \mathcal{R})$ is isomorphic to a subset of $\cup_{\alpha \in I} \mathsf{chase}(\{\alpha\}, \mathcal{R})$. In the other direction, the sequence of triggers in a derivation from any atom $\alpha \in I$ defines a derivation from I. It follows that $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ is isomorphic to a subset of $\mathsf{chase}(I, \mathcal{R})$. However, while $\cup_{\alpha \in I} \mathsf{chase}(\{\alpha\}, \mathcal{R})$ and $\mathsf{chase}(I, \mathcal{R})$ are homomorphically equivalent, they are not necessarily isomorphic, as illustrated by the next example.

Example 2. *Let* $I = \{p(a, b), q(a, b)\}$ *and* $\mathcal{R} = \{\rho_1, \rho_2\}$ *with* $\rho_1 = p(x, y) \rightarrow q(x, y)$ *and* $\rho_2 = q(x, y) \rightarrow \exists z\, r(y, z)$. *Then* $\mathsf{chase}(I, \mathcal{R}) = \{p(a, b), q(a, b), r(b, z_0)\}$, *where* z_0 *is a null, by the sequence of triggers* $\mathsf{t}_1 = (\rho_1, \{x \mapsto a, y \mapsto b\})$, $\mathsf{t}_2 = (\rho_2, \{x \mapsto a, y \mapsto b\})$ *(or the reverse sequence). Trigger* t_1 *does not generate any atom because it produces* $q(a, b)$ *and* $q(a, b) \in I$. *Let* $\alpha_1 = p(a, b) \in I$: $\mathsf{chase}(\{\alpha_1\}, \mathcal{R}) = \{p(a, b), q(a, b), r(b, z_0)\}$ *by the same sequence of triggers. Now,* t_1 generates *the atom* $q(a, b)$. *Let* $\alpha_2 = q(a, b) \in I$: $\mathsf{chase}(\{\alpha_2\}, \mathcal{R}) = \{q(a, b), q(b, z_1)\}$ *by* t_2. *The union of the two chases is:*

$$\{p(a, b), q(a, b), r(b, z_0), r(b, z_1)\}$$

which maps to $\mathsf{chase}(I, \mathcal{R})$ *by the homomorphism* $\pi = \{z_1 \mapsto z_0\}$, *but is not isomorphic to it.*

Let us say that an atom α_j is *generated* from an atom α_i in $\mathsf{chase}(I, \mathcal{R})$ if *(i)* α_j is directly generated by a trigger on α_i, or *(ii)* α_j is directly generated by a trigger on an atom itself generated from α_i. In the rest of this paper, we will specifically rely on the following two immediate properties. First, atoms generated from distinct (ground) atoms α and α' in I do not share any null (Proposition 3). Second, the set of atoms A_α generated from an atom $\alpha \in I$ forms a subset of $\mathsf{chase}(\{\alpha\}, \mathcal{R})$, up to the choice of nulls (Proposition 4). We will in fact only use a corollary of Proposition 4: every finite set of atoms that are all generated from an atom $\alpha \in I$ forms a subset of $\mathsf{chase}(\{\alpha\}, \mathcal{R})$.

Proposition 3. *Given any* $\mathcal{K} = (I, \mathcal{R})$, *let* α_i *and* α_j *be generated from distinct atoms in* I *by* $\mathsf{chase}(I, \mathcal{R})$. *Then,* $\mathsf{vars}(\alpha_i) \cap \mathsf{vars}(\alpha_j) = \emptyset$.

Proof. If two (distinct) atoms α_i and α_j share a (necessarily fresh) variable, then they are generated from the same ground atom from I. Indeed, let α_k be the atom in which this variable is introduced. If $\alpha_k = \alpha_i$ or $\alpha_k = \alpha_j$, then one atom is generated from the other. Otherwise, α_i and α_j are both generated from α_k. In all the cases, α_i and α_j are generated from the same atom $\alpha \in I$ from which α_k is generated. □

Proposition 4. *For any KB $\mathcal{K} = (I, \mathcal{R})$ with $\mathcal{R}$ a linear ruleset, atom $\alpha \in I$ and set A_α of all the atoms generated from α in* $\mathsf{chase}(I, \mathcal{R})$*, there is an isomorphism from $\{\alpha\} \cup A_\alpha$ to a subset of* $\mathsf{chase}(\{\alpha\}, \mathcal{R})$.

Proof. Let $D_\alpha = I_0(=\alpha)\ (\rho_1, \pi_1)\ I_1 \ldots\ (\rho_n, \pi_n)\ I_n, \ldots$ be the (possibly infinite) derivation resulting in $\{\alpha\} \cup A_\alpha$, which is extracted from the derivation associated with $\mathsf{chase}(I, \mathcal{R})$. We inductively build a sequence of substitutions $\phi_0, \ldots, \phi_n, \ldots$, such that for all $i \geq 0$, ϕ_i is an isomorphism from I_i to a subset of $\mathsf{chase}(I, \mathcal{R})$. For $i = 0$, ϕ_i is the empty substitution (i.e., $\phi_0(\alpha) = \alpha$). For $i > 0$, let $\alpha_i = I_i \setminus I_{i-1}$, and let $t_i = (\rho_i, \pi_i)$ be the trigger that produces α_i. Let α_{j_i} be the atom on which t_i is applied. If α_i does not contain a new null, $\phi_i = \phi_{i-1}$. Otherwise, let $\alpha' = \phi_{i-1}(\alpha_{j_i})$: the pair $t' = (\rho_i, \phi_{i-1} \circ \pi_i)$ is a trigger on α' and it necessarily *generates* an atom since a new null is created. Then, ϕ_i is obtained by extending ϕ_{i-1} in the obvious way to bijectively map the nulls introduced by t_i to those introduced by t'. By construction, ϕ_i is an isomorphism from I_i to a subset of $\mathsf{chase}(\{\alpha\}, \mathcal{R})$. Furthermore, $\cup_{i=0}^{\infty} \phi_i$ is an injective substitution of $\mathsf{vars}(\{\alpha\} \cup A_\alpha)$ by $\mathsf{vars}(\mathsf{chase}(\{\alpha\}, \mathcal{R}))$, i.e., an isomorphism from $\{\alpha\} \cup A_\alpha$ to a subset of $\mathsf{chase}(\{\alpha\}, \mathcal{R})$. □

2.3 Queries

We now define the classes of queries we consider: conjunctive queries, regular path queries and their common generalisation, namely conjunctive regular path queries. In short, a conjunctive query is an existentially quantified conjunction of standard atoms; a regular path query is a single "path atom", i.e., an atom on a binary predicate corresponding to a regular language; and a conjunctive regular path query is an existentially quantified conjunction of standard and path atoms.

A regular language can be represented by a regular expression or by a non-deterministic finite automaton (NFA). Let Σ be a finite set of symbols. A regular expression $\mathcal{E}$ over Σ is defined by the grammar: $\mathcal{E} \rightarrow \varepsilon \mid a \mid \mathcal{E} \cdot \mathcal{E} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E}^*$, where $a \in \Sigma$ and ε denotes the empty word. 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. For convenience, we sometimes use the notation $s' \in \delta(s, a)$ to denote that $(s, a, s') \in \delta$. Since NFAs are exponentially more succinct than regular expressions (Ehrenfeucht and Zeiger 1976), the complexity proofs should use the NFA representation for upper bounds and the regular expression representation for lower bounds, so that the results hold regardless of the representation. We indeed use the NFA representation for upper bounds. For lower bounds, we mostly rely on previous results, which indeed use the regular expression representation; for our only proof of a lower bound (Section 4), the query is of the form p^*, hence the choice of a representation does not matter.

We denote by $\mathcal{L}(\mathcal{E})$ the language defined by the regular expression $\mathcal{E}$. Simarly, we denote by $\mathcal{L}(\mathbb{A})$ the language recognized by the automaton $\mathbb{A}$. Furthermore, we denote by $\mathcal{L}_{\mathbb{A}}(s_i, s_j)$ the set of words that take $\mathbb{A}$ to the state s_j when they are read from the state s_i. When $s_i = s_0$ and $s_j \in F$, $\mathcal{L}_{\mathbb{A}}(s_i, s_j) \subseteq \mathcal{L}(\mathbb{A})$.

We denote by $\mathcal{P}_2$ the subset of binary predicates in the considered vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$. Given a binary predicate r, we also consider its inverse predicate r^- (i.e., for all terms t_1 and t_2, $r(t_1, t_2)$ holds iff $r^-(t_2, t_1)$ holds). We set $\mathcal{P}_2^{\pm} = \mathcal{P}_2 \cup \{r^- \mid r \in \mathcal{P}_2\}$. A *path predicate* Λ is a binary predicate given by an NFA or a regular expression defining a regular language $L(\Lambda)$ over $\mathcal{P}_2^{\pm}$. A *path atom* is an atom built on a path predicate, i.e., it takes the form $\Lambda(t_1, t_2)$, where Λ is a path predicate and t_1, t_2 are terms. Note that standard binary predicates are a special case of path predicates. We assume without loss of generality that automata associated with distinct path predicates have disjoint sets of states.

A *conjunctive (two-way) regular path query* (CRPQ)[2] has the form $q(\mathbf{x}) = \exists \mathbf{y} B$, where $\mathbf{x}$ and $\mathbf{y}$ are disjoint tuples of variables, and B is a conjunction of standard and path atoms with $\mathsf{terms}(B) = \mathbf{x} \cup \mathbf{y}$. The free variables of q, i.e., the $\mathbf{x}$ variables, are called *answer variables*. A CRPQ is *Boolean* if it is a closed formula, i.e., $\mathbf{x} = \emptyset$. A *conjunctive query* (CQ) is a CRPQ $q(\mathbf{x}) = \exists \mathbf{y} B$ where B contains only standard atoms. A *regular path query* (RPQ) is a CRPQ

[2] As we only consider the two-way variant, we will use the abbreviation (C)RPQ instead of the more usual (C)2RPQ.

of the form $q(x_1, x_2) = \Lambda(x_1, x_2)$, where $\Lambda(x_1, x_2)$ is a path atom. Hence, an RPQ contains exactly two variables, which are answer variables.

Given an interpretation $\mathcal{I} = (\Delta, .^{\mathcal{I}})$, we extend the set of interpreted symbols as follows. First, for every $r \in \mathcal{P}_2$, we set $(r^-)^{\mathcal{I}} = \{(e_2, e_1) | (e_1, e_2) \in r^{\mathcal{I}}\}$. Second, we call *path* (from e_0 to e_n) in $\mathcal{I}$ a (finite) sequence $e_0 r_1 e_1 \dots r_n e_n$, with $n \geq 0$ such that $e_0 \in \Delta_{\mathcal{I}}$ and for every $1 \leq i \leq n$, $e_i \in \Delta_{\mathcal{I}}$, $r_i \in \mathcal{P}_2^{\pm}$ and $(e_{i-1}, e_i) \in r_i^{\mathcal{I}}$. The *label* $\lambda(p)$ of a path $p = e_0 r_1 e_1 \dots r_n e_n$ is the word $r_1 \dots r_n$. If $n = 0$, $\lambda(p)$ is the empty word ε. Then, for any path predicate Λ, we set:

$$\Lambda^{\mathcal{I}} = \{(e_0, e_n) | \text{ there is a path } p \text{ from } e_0 \text{ to } e_n \text{ in } \mathcal{I} \text{ such that } \lambda(p) \in \mathcal{L}(\Lambda)\}$$

The notion of model of a Boolean CRPQ is the classical logical notion, up to the above extensions of the set of interpreted symbols. The notion of match in an interpretation is extended in the same way, i.e., a *match* of a CRPQ q in an interpretation $\mathcal{I}$ is a mapping π from terms(q) to elements of $\Delta_{\mathcal{I}}$ such that *(i)* $\pi(a) = a^{\mathcal{I}}$ for each constant a; *(ii)* $\pi(\mathbf{t}) \in r^{\mathcal{I}}$ for each (standard or path) atom $r(\mathbf{t})$ in q. It follows that an interpretation $\mathcal{I}$ is a model of a Boolean CRPQ q iff there is a *match* of q in $\mathcal{I}$.

Given a CRPQ $q(\mathbf{x}) = \exists \mathbf{y} B$ with $\mathbf{x} = (x_1 \dots x_k)$, a tuple of constants $\mathbf{a} = (a_1 \dots a_k)$ is an *answer* to q in $\mathcal{I}$ if there is a match π of q in $\mathcal{I}$ such that $\pi(x_i) = a_i^{\mathcal{I}}$ for $1 \leq i \leq k$. Equivalently, $\mathcal{I}$ is a model of the Boolean CRPQ $q()$ obtained from $q(\mathbf{x})$ by substituting each x_i with a_i.

We can now define the notion of a certain answer to a CRPQ on a KB. Given a CRPQ $q(\mathbf{x}) = \exists \mathbf{y} B$ and a KB $\mathcal{K} = (I, \mathcal{R})$, a tuple of constants $\mathbf{a} = s(\mathbf{x})$, where s denotes a substitution, is a *certain answer* to a CRPQ $q(\mathbf{x})$ over $\mathcal{K}$ if $I \cup \mathcal{R} \models q(\mathbf{a})$, where $q(\mathbf{a}) = \exists \mathbf{y}.s(B)$. In other words, $\mathbf{a}$ is an answer to q in each model of $\mathcal{K}$. Concerning the specific case of q being a Boolean CRPQ, we have that the empty tuple () is a certain answer to q on $\mathcal{K}$ if $I \cup \mathcal{R} \models q$.

Although certain answers are defined with respect to all models of the KB, it is actually sufficient to consider answers in the unique model defined by the chase of the KB. This immediately follows from two properties: *(i)* the chase of a KB $\mathcal{K}$ is a universal model of $\mathcal{K}$, and *(ii)* CRPQs are closed under homomorphism, which means that for any Boolean CRPQ q and model $\mathcal{I}$ of q, any interpretation $\mathcal{I}'$ to which $\mathcal{I}$ homomorphically maps is also a model of q.

Proposition 5. *For any KB $\mathcal{K}$ and CRPQ $q(\mathbf{x})$, a tuple of constants $(\mathbf{a})$ is a certain answer to q over $\mathcal{K}$ if and only if there is a match π of q in chase($\mathcal{K}$) such that $\pi(\mathbf{x}) = (\mathbf{a})$.*

In the following, we will identify the chase with the interpretation naturally associated with it. In particular, a *path of terms* in the chase is the natural translation of a path in the associated interpretation. We denote a path of terms by $p = t_0 r_1 t_1 \cdots r_n t_n$, where the t_i are terms, and the r_i elements of $\mathcal{P}_2^{\pm}$, and simply by $t_0 \cdots t_n$ when only the extremities are needed.

Finally, we point out that, whenever there is a match of a CRPQ in the possibly infinite chase, this match can be found in a finite portion of the chase.

Proposition 6. *For any KB $\mathcal{K} = (I, \mathcal{R})$ and CRPQ q, if there a match π of q in* chase$(\mathcal{K})$*, then for any chase sequence $D = I_0 (= I)\ I_1 \dots\ I_n, \dots$ of $\mathcal{K}$, there is I_i in D such that π is a match of q in I_i.*

Proof. (Sketch) Let π be a match of q in chase($\mathcal{K}$). Let $D = I_0 (= I)\ I_1 \dots\ I_n, \dots$ be a chase sequence of $\mathcal{K}$. Given any standard atom α in q, let rank(α) be the smallest i such that $\pi(\alpha) \in I_i$. Given any path atom $\alpha = \Lambda(t, t')$ in q, let rank(α) be the smallest i such that I_i contains all the (standard) atoms of a path $t_0 (= \pi(t)) r_1 t_1 \cdots r_n t_n (= \pi(t'))$. Let j be the maximal value of rank(α) among all standard and path atoms α in q. Then, π is a match of q in I_j. □

Hence, given a Boolean CRPQ q and a KB $\mathcal{K} = (I, \mathcal{R})$, it holds that $\mathcal{K} \models q$ if and only if there a finite $\mathcal{R}$-derivation from I that results in a set of atoms I_n such that $I_n \models q$.

2.4 Query Answering Problem

Let $\mathbf{C} \in \{\text{CQ, RPQ, CRPQ}\}$ denote a class of queries. The $\mathbf{C}$ *answering problem* asks, given a KB $\mathcal{K} = (I, \mathcal{R})$, a query q from the class $\mathbf{C}$ and a tuple of constants $\mathbf{a}$, whether $\mathbf{a}$ is an answer to q over $\mathcal{K}$. Since ground atom entailment from an existential rule KB is undecidable in general (see (Beeri and Vardi 1981, Chandra et al. 1981) for the first undecidability results about equivalent problems on tuple-generating dependencies), the $\mathbf{C}$ answering problem is undecidable for any query class $\mathbf{C}$.

We will study the complexity of (C)RPQ answering in the case of linear and guarded rules, according to two complexity measures: *combined* complexity, where $\mathcal{K}$ and q are both part of the problem input, and *data* complexity, where q and $\mathcal{R}$ are fixed, and only I is part of the input. This second complexity measure is specially relevant when it can be assumed that the sizes of the query and the set of rules are small compared to the size of the instance. With respect to combined complexity, we will in turn distinguish between two cases, depending on whether the predicate arity is *bounded* or *unbounded*.

We will refer to standard complexity classes: NL (problems solvable in non-deterministic logarithmic space), also called NLogSpace, PTime (problems solvable in deterministic polynomial time), NP (problems solvable in non-deterministic polynomial time), PSpace (problems solvable in polynomial space), ExpTime (problems solvable in deterministic exponential time) and 2ExpTime (problems solvable in deterministic doubly exponential time). In Table 1, we furthermore mention AC_0, a class from circuit complexity, which is a strict subclass of NL. To sum up, these classes are ordered as follows: $\text{AC}_0 \subset \text{NL} \subseteq \text{PTime} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime} \subseteq \text{2ExpTime}$.

The complexities of CQ answering under linear or guarded rules are recalled in Tables 1 and 2. It is well known that CQ evaluation (or CQ answering with $\mathcal{R} = \emptyset$) is in AC_0 for data complexity and NP-complete for combined complexity, regardless of the assumption on predicate arity. Under linear rules, CQ answering remains in AC_0 for data complexity (Calì et al. 2009b) and NP-complete for combined complexity with bounded arity (follows from (Johnson and Klug 1984, Gottlob and Schwentick 2012))[3] but it becomes PSpace-complete for combined complexity with unbounded arity (Calì et al. 2009a). Under guarded rules, CQ answering is PTime-complete for data complexity (Calì et al. 2009b), ExpTime-complete for combined complexity with bounded arity (Calì et al. 2013) and 2ExpTime-complete for combined complexity with unbounded arity (Calì et al. 2013).

For the sake of comparison, let us further recall the complexity[4] of answering (C)RPQs over graph databases, in the absence of any ontology. RPQ evaluation is known to be NL-complete in both data and combined complexity. For CRPQs, the data complexity remains NL-complete, but the combined complexity rises to NP-complete (matching that of CQ evaluation in databases).

To prove the results about CRPQ answering, it will be convenient to consider the following problem, called *CRPQ Entailment*, which asks, given a KB $\mathcal{K} = (I, \mathcal{R})$ and a Boolean CRPQ q, if $I \cup \mathcal{R} \models q$. Both problems are linearly reducible one to the other, since, given a Boolean CRPQ q, it holds that $I \cup \mathcal{R} \models q$ if and only if $()$ is a certain answer to q, and, in turn, a tuple of constants $(\mathbf{a})$ is a certain answer to a CRPQ $q(\mathbf{x})$ if and only if $I \cup \mathcal{R} \models q_{(\mathbf{x} \mapsto \mathbf{a})}$, where $q_{(\mathbf{x} \mapsto \mathbf{a})}$ is the Boolean CRPQ obtained from $q(\mathbf{x})$ by substituting each x_i in $\mathbf{x}$ with a_i in $\mathbf{a}$.

[3] (Johnson and Klug 1984) proves that inclusion dependencies (a subclass of linear rules) of bounded predicate arity enjoy the Polynomial Witness Property and (Gottlob and Schwentick 2012) points out that the proof extends to linear rules. This implies the NP membership. NP-hardness follows from the complexity of CQ evaluation.

[4] The reason that we do not provide references for the complexity results for graph databases is that they are considered folklore in the area (and explicitly noted as such in the survey chapter (Figueira 2021)).

3 RPQ Answering under Linear Rules: Upper Bound

We consider the problem of computing the certain answers to a regular path query in the presence of a set of linear rules and present an algorithm that is inspired by a related algorithm for DL-Lite ontologies (Bienvenu et al. 2015) (see Section 7 for discussion). The algorithm takes as input an RPQ whose regular language is given by an NFA and works roughly as follows:

- 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 passes through nulls in the chase, these terms are not guessed, instead the state of the automaton when the path returns to constants of the instance is guessed.

The first item corresponds to a standard algorithm for RPQ answering over bare data. The real difficulty lies in the second item, which requires a characterization of when it is possible to reach an automaton state through a path involving null elements.

The following proposition makes a first step towards such a characterization by observing that every 'anonymous' path in the chase can be localized within the chase of a single ground atom. It exploits the fact that for linear rules, each atom can be 'chased' independently (see Section 2.2).

Proposition 7. *Let $\mathcal{R}$ be a set of linear rules and I be an instance. Then for every path $p = d_0 r_1 d_1 \cdots r_n d_n$ in $\mathsf{chase}(I, \mathcal{R})$ such that $d_0, d_n \in \mathsf{terms}(I)$ and $d_i \notin \mathsf{terms}(I)$ for $0 < i < n$, there exists an atom $\alpha \in I$ with $d_0, d_n \in \mathsf{terms}(\alpha)$ such that there is a path p' in $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ from d_0 to d_n with $\lambda(p') = \lambda(p)$.*

Proof. Let A_p be the set of atoms in p. Any consecutive atoms in A_p, i.e., $r_i(d_{i-1}, d_i)$ and $r_{i+1}(d_i, d_{i+1})$, $0 < i < n$, share a null, hence, by Proposition 3, there is a unique atom $\alpha \in I$ from which all the atoms in A_p are generated in $\mathsf{chase}(I, \mathcal{R})$. From Proposition 4, A_p is isomorphic to a subset of $\mathsf{chase}(\{\alpha\}, \mathcal{R})$, which yields the path p'. □

We can therefore concentrate on identifying the paths that occur within the chase of a single atom. In order to abstract from the particular terms occurring in an atom, we introduce the following notion of type:

Definition 8 (Type of an atom). *A* type *is a pair $T = (r, \sim_T)$ where r is a predicate of arity k and $\sim_T$ is an equivalence relation on $\{1, \ldots, k\}$. The type of an atom α, denoted by* $\mathsf{type}(\alpha)$*, is the pair $T = (r, \sim_T)$ where r is the predicate of α and $i \sim_T j$ iff the i^{th} and the j^{th} arguments of α are equal.*

For instance, the atoms $r(x, y)$, $r(y, a)$ and $r(a, b)$, where a and b are constants, have the same type $T = (r, \{\{1\}, \{2\}\})$, while the atom $r(x, x)$ has type $(r, \{\{1, 2\}\})$; here, $\sim_T$ is given by its set of equivalence classes.

When atoms α_1 and α_2 have the same type, there exists a bijective mapping θ_{12} from $\mathsf{terms}(\alpha_1)$ to $\mathsf{terms}(\alpha_2)$ such that $\alpha_2 = \theta_{12}(\alpha_1)$. We call this mapping the *natural mapping* from α_1 to α_2 and observe that $\theta_{21} = \theta_{12}^{-1}$. With this notion in hand, we can formalize the idea that atoms with the same type behave the same regarding rule applications.

Proposition 9. *Let α_1 and α_2 be two atoms of the same type, and let θ_{12} be the natural mapping from α_1 to α_2. If $\rho \in \mathcal{R}$ is applicable to α_1 by π creating α_1', then ρ is applicable to α_2 by $\theta_{12} \circ \pi$ creating α_2', where α_1' and α_2' have the same type, and the natural mapping θ_{12}' from α_1' to α_2' coincides with θ_{12} on the terms on which they are both defined.*

A corollary of this proposition is that $\mathsf{chase}(\{\alpha_1\}, \mathcal{R})$ and $\mathsf{chase}(\{\alpha_2\}, \mathcal{R})$ will contain the same kinds of paths, as formalized next.

Corollary 10. *Let α and α' be atoms with $\mathsf{type}(\alpha) = \mathsf{type}(\alpha')$. Then there is a path in $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ from term $\alpha[i]$ to term $\alpha[j]$ with label l if and only if there is a path in $\mathsf{chase}(\{\alpha'\}, \mathcal{R})$ from $\alpha'[i]$ to $\alpha'[j]$ with label l.*

Proof. From Proposition 9, there is a bijective mapping θ from the terms in $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ to the terms in $\mathsf{chase}(\{\alpha'\}, \mathcal{R})$, such that $\theta(\mathsf{chase}(\{\alpha\}, \mathcal{R})) = \mathsf{chase}(\{\alpha'\}, \mathcal{R})$. Hence, every path p in $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ from term $\alpha[i]$ to term $\alpha[j]$ is mapped by θ to a path p' in $\mathsf{chase}(\{\alpha'\}, \mathcal{R})$ from $\alpha'[i]$ to $\alpha'[j]$ with the same label as p. □

The following proposition shows that it is possible to take paths in the chase of a single atom and reproduce them in a larger chase that contains an atom of the same type.

Proposition 11. *Let α, α', β be atoms such that* $\mathsf{type}(\alpha) = \mathsf{type}(\alpha')$ *and* $\alpha \in \mathsf{chase}(\{\beta\}, \mathcal{R})$. *If there is a path in* $\mathsf{chase}(\{\alpha'\}, \mathcal{R})$ *from term $\alpha'[i]$ to term $\alpha'[j]$ with label l, then there is a path in* $\mathsf{chase}(\{\beta\}, \mathcal{R})$ *from $\alpha[i]$ to $\alpha[j]$ with label l.*

Proof. Let p' be a path in $\mathsf{chase}(\{\alpha'\}, \mathcal{R})$ from term $\alpha'[i]$ to term $\alpha'[j]$ with label l. Since $\mathsf{type}(\alpha) = \mathsf{type}(\alpha')$, by Corollary 10, there is a path p in $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ from term $\alpha[i]$ to term $\alpha[j]$ with label l. We then show that $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ is isomorphic to a subset of $\mathsf{chase}(\{\beta\}, \mathcal{R})$, from which we conclude that $\mathsf{chase}(\{\beta\}, \mathcal{R})$ contains a path p'' from $\alpha[i]$ to $\alpha[j]$ with label l. We prove that there is an isomorphism from the result of any $\mathcal{R}$-derivation from α to a subset of $\mathsf{chase}(\{\beta\}, \mathcal{R})$, by induction on the length of this derivation. At rank 0, the property is true since $\alpha \in \mathsf{chase}(\{\beta\}, \mathcal{R})$. Assume it is true until rank n. Consider a derivation from α of length $n+1$: let A_n be the set of atoms obtained after applying the n^{th} trigger; let (ρ, π) be the $n^{\text{th}}+1$ trigger, $a = \pi(\mathsf{body}(\rho))$ and a' be the resulting atom. Let ψ_n be the isomorphism from A_n to a subset of $\mathsf{chase}(\{\beta\}, \mathcal{R})$. Since a and $\psi_n(a)$ have the same type, $(\rho, \psi_n \circ \pi)$ is a trigger on $\psi_n(a)$ and occurs in the derivation that builds $\mathsf{chase}(\{\beta\}, \mathcal{R})$, creating atom a'' of the same type as a' (as in Proposition 9). We build the mapping ψ_{n+1} from $A_n \cup \{a'\}$ to $\mathsf{chase}(\{\beta\}, \mathcal{R})$ by extending ψ_n according to the natural mapping from a' to a''. Since ψ_n is an isomorphism from A_n to a subset of $\mathsf{chase}(\{\beta\}, \mathcal{R})$, ψ_{n+1} is an isomorphism from $A_n \cup \{a'\}$ to a subset of $\mathsf{chase}(\{\beta\}, \mathcal{R})$. □

It now remains to determine which paths are available starting from an atom of a given type. More specifically, we want to know, given a type T, positions j, j' in any atom α of type T, automaton $\mathbb{A}$, and states s_i, s_i', whether $\mathsf{chase}(\alpha, \mathcal{R})$ contains a path p from $\alpha[j]$ to $\alpha[j']$ whose label $\lambda(p)$ takes $\mathbb{A}$ from state s_i to state s_i'. To this end, we formalize the notion of a type admitting a loop:

Definition 12. *We say that a* type T admits an (s_i, j, s_i', j')-loop (w.r.t. a linear ruleset $\mathcal{R}$ and NFA $\mathbb{A}$) *if there is a path p in* $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ *from $\alpha_T[j]$ to $\alpha_T[j']$ such that $\lambda(p) \in \mathcal{L}_{\mathbb{A}}(s_i, s_{i'})$, where α_T is any atom of type T.*

Any choice of the atom α_T will give the same result, due to Corollary 10. Before detailing the algorithms, we provide below a high-level overview of how they work using the running example.

Example 13 (Running example—continued). *Let $\mathcal{R}$ be the following set of linear rules:*

(ρ_1) $\mathsf{isFriendOf}(x, y) \rightarrow \mathsf{isFriendOf}(y, x)$
(ρ_2) $\mathsf{isFriendOf}(x, y) \rightarrow \mathsf{follows}(x, y)$
(ρ_3) $\mathsf{follows}(x, y) \rightarrow \exists m\, \mathsf{message}(m, x, y)$
(ρ_5) $\mathsf{message}(m, x, y) \rightarrow \mathsf{sends}(x, m)$
(ρ_6) $\mathsf{message}(m, x, y) \rightarrow \mathsf{receives}(y, m)$

The first three rules were already introduced in Example 1 and the two additional rules allow us to reformulate the CRPQ q_6 in the form of an RPQ. Recall that q_6 asks whether user Alice *(abbreviated as* A *in the following) received a message from someone she follows directly or indirectly. Instead of the CRPQ q_6, we can consider the following RPQ q_6' and check whether* (A, A) *is a certain answer:*

$$q_6'(x, y) = \mathsf{follows} \cdot \mathsf{follows}^* \cdot \mathsf{sends} \cdot \mathsf{receives}^-(x, y)$$

We assign to q_6' the NFA representation $\mathbb{A}$ with states $\{s_0, s_1, s_2, s_f\}$ (with s_0 initial and s_f final) and four transitions: $(s_0, \mathsf{follows}, s_1)$, $(s_1, \mathsf{follows}, s_1)$, $(s_1, \mathsf{sends}, s_2)$ *and* $(s_2, \mathsf{receives}^-, s_f)$.

Let us consider again the instance $I = \{\mathsf{follows}(\mathsf{B},\mathsf{A}), \mathsf{isFriendOf}(\mathsf{C},\mathsf{A}), \mathsf{isFriendOf}(\mathsf{C},\mathsf{B})\}$. *Query* q_6' *can be matched to the following path of terms in* $\mathsf{chase}(I,\mathcal{R})$*:*

$$(\mathsf{A}\ \mathsf{follows}\ \mathsf{C}\ \mathsf{follows}\ \mathsf{B}\ \mathsf{sends}\ \mathsf{m}_0\ \mathsf{receives}^-\mathsf{A})$$

with m_0 *the null introduced when applying the rule* (ρ_3) *to* $\mathsf{follows}(\mathsf{B},\mathsf{A})$. *The facts* $\mathsf{follows}(\mathsf{A},\mathsf{C})$ *and* $\mathsf{follows}(\mathsf{C},\mathsf{B})$, *witnessing the subpath* $(\mathsf{A}\ \mathsf{follows}\ \mathsf{C}\ \mathsf{follows}\ \mathsf{B})$, *are derived from the facts* $\mathsf{isFriendOf}(\mathsf{C},\mathsf{A})$ *and* $\mathsf{isFriendOf}(\mathsf{C},\mathsf{B})$, *respectively, using rules* ρ_1 *and* ρ_2. *As these are ground facts, the corresponding subpath* $(A\ \mathsf{follows}\ C\ \mathsf{follows}\ B)$ *will be entirely guessed by the algorithm. This subpath takes* $\mathbb{A}$ *from state* s_0 *to* s_1. *By contrast, the subpath* $(B\ \mathsf{sends}\ m_0\ \mathsf{receives}^-A)$, *whose witnessing atoms contain the null* m_0, *will not be guessed. Instead, we will rely on Proposition 7 and find a ground fact in* I, *here* $\alpha = \mathsf{follows}(B,A)$, *such that* $(B\ \mathsf{sends}\ m_0\ \mathsf{receives}^-A)$ *occurs in* $\mathsf{chase}(\alpha,\mathcal{R})$. *More precisely: the type* $(\mathsf{follows}, \{\{1\},\{2\}\})$, *which is the type of* α, *admits a loop from the term in position 1 to the term in position 2 that takes* $\mathbb{A}$ *from state* s_1 *to* s_f *(in short: an* $(s_1, 1, s_f, 2)$*-loop).*

We will use Algorithm 1, detailed later, to compute the set of loops admitted by each of the types associated with $\mathcal{R}$. *In particular, we have here that* (i) *type* $(\mathsf{sends}, \{\{1\},\{2\}\})$ *admits an* $(s_1, 1, s_2, 2)$*-loop and* (ii) *type* $(\mathsf{receives}^-, \{\{1\},\{2\}\})$ *admits an* $(s_2, 2, s_f, 1)$*-loop, which is directly obtained from the transitions in* $\mathbb{A}$. *From* (i) *and rule* (ρ_5) *considered "from head to body", we obtain that type* $(\mathsf{message}, \{\{1\},\{2\},\{3\}\})$ *admits an* $(s_1, 2, s_2, 1)$*-loop. Similarly, from* (ii) *and rule* (ρ_6), *we obtain that type* $(\mathsf{message}, \{\{1\},\{2\},\{3\}\})$ *admits an* $(s_2, 1, s_f, 3)$*-loop. Concatenating the two loops admitted by type* $(\mathsf{message}, \{\{1\},\{2\},\{3\}\})$, *we add an* $(s_1, 2, s_f, 3)$*-loop to that type, which, using rule* (ρ_3), *yields the* $(s_1, 1, s_f, 2)$*-loop for the type* $(\mathsf{follows}, \{\{1\},\{2\}\})$ *of* $\alpha = \mathsf{follows}(B,A)$.

Algorithm 2, which is the global algorithm, takes as input $I, \mathcal{R}, \mathbb{A}$, *as well as a candidate answer (the pair* (A,A) *in our example). It first calls Algorithm 1 to build the table of loops associated with types, then tries to verify the existence of a witnessing path in* $\mathsf{chase}(I,\mathcal{R})$, *while being guided by the transitions in* $\mathbb{A}$ *and loops assigned to types. Here, starting from state* s_0 *and constant* A, *it will guess the atoms* $\mathsf{follows}(\mathsf{A},\mathsf{C})$ *and* $\mathsf{follows}(\mathsf{C},\mathsf{B})$ *using transitions in* $\mathbb{A}$ *and check that these atoms are indeed entailed by* $(I,\mathcal{R})$. *This corresponds to the subpath* $(A\ \mathsf{follows}\ C\ \mathsf{follows}\ B)$, *which takes* $\mathbb{A}$ *from state* s_0 *to* s_1. *Then, the algorithm guesses constant* A *and state* s_f *together with the* $(s_1, 1, s_f, 2)$*-loop for type* $(\mathsf{follows}, \{\{1\},\{2\}\})$, *and checks that* I *contains the atom* $\mathsf{follows}(B,A)$.

We shall now explain how to compute the loops admitted by a given type. Algorithm 1 builds a table Loop whose cells are indexed by tuples (s_i, j, s_i', j'), with the cell (s_i, j, s_i', j') storing the types T that admit an (s_i, j, s_i', j')-loop. We use the notation $\mathsf{Loop}(s_i, j, s_i', j')$ to denote the set of types stored in the cell (s_i, j, s_i', j'). We assume all the cells are empty when they are created. Line 4 initializes the table by stating that one can go from a term in a position to the same term in another position without reading any word (and thus not moving in the automaton). Lines 7 and 9 correspond to traversing a single binary atom, reading its label either as an r or an r^-, in the case where both terms are distinct, i.e., $T = (r, \{\{1\},\{2\}\})$. Lines 12 to 15 are similar but handle the case where both arguments are equal, i.e., $T = (r, \{\{1,2\}\})$. Finally, Lines 18 and 21 serve to combine these basic paths. On the one hand, the table is saturated through path concatenation: if one can go from a position j_1 to a position j_2 by a path taking the automaton from state s_1 to state s_2, and from j_2 to a position j_3 by a path taking the automaton from s_2 to state s_3, then one can go from j_1 to j_3 by a path taking the automaton from s_1 to s_3. On the other hand, rules are taken into account: for any rule $\alpha \to \beta$, if α and β both contain variables x and y (with possibly $x = y$) and one can go from the position of x in β to the position of y in β taking the automaton from state s_1 to state s_2 then the same move is possible from the position of x in α to the position of y in α. Indeed, in the chase of any atom α_1 of type $\mathsf{type}(\alpha)$, there is an atom β_1 of type $\mathsf{type}(\beta)$, which has the same terms as α_1 in the positions of x and y. Next Example 14 further illustrates the different steps of Algorithm 1.

Algorithm 1: Creating the Loop table

Data: A set of linear rules $\mathcal{R}$ and NFA $\mathbb{A} = (S, \Sigma, \delta, s_0, F)$
Result: A table whose cells are indexed by tuples $(s, j, s', j') \in S \times [1, w] \times S \times [1, w]$, with w the maximal predicate arity in $\mathcal{R}$, and each cell contains a set of types on predicates from $\mathcal{R}$

/* Initialization step */
for *type* T *of predicate of arity* k **do**
  for *pair* $(j, j') \in \{1, \dots, k\}^2$ *with* $j \sim_T j'$ **do**
    for $s \in S$ **do**
      $\mathsf{Loop}(s, j, s, j') \leftarrow \mathsf{Loop}(s, j, s, j') \cup \{T\}$;
for *type* T *based on* $r(x, y)$ **do**
  for $(s_1, r, s_2) \in \delta$ **do**
    $\mathsf{Loop}(s_1, 1, s_2, 2) \leftarrow \mathsf{Loop}(s_1, 1, s_2, 2) \cup \{T\}$;
  for $(s_1, r^-, s_2) \in \delta$ **do**
    $\mathsf{Loop}(s_1, 2, s_2, 1) \leftarrow \mathsf{Loop}(s_1, 2, s_2, 1) \cup \{T\}$;
for *type* T *based on* $r(x, x)$ **do**
  for $(s_1, r, s_2) \in \delta$ *or* $(s_1, r^-, s_2) \in \delta$ **do**
    $\mathsf{Loop}(s_1, 1, s_2, 1) \leftarrow \mathsf{Loop}(s_1, 1, s_2, 1) \cup \{T\}$;
    $\mathsf{Loop}(s_1, 1, s_2, 2) \leftarrow \mathsf{Loop}(s_1, 1, s_2, 2) \cup \{T\}$;
    $\mathsf{Loop}(s_1, 2, s_2, 1) \leftarrow \mathsf{Loop}(s_1, 2, s_2, 1) \cup \{T\}$;
    $\mathsf{Loop}(s_1, 2, s_2, 2) \leftarrow \mathsf{Loop}(s_1, 2, s_2, 2) \cup \{T\}$;
/* Saturation step */
while *something added* **do**
  for $T \in \mathsf{Loop}(s_1, j_1, s_2, j_2) \cap \mathsf{Loop}(s_2, j_2, s_3, j_3)$ **do**
    $\mathsf{Loop}(s_1, j_1, s_3, j_3) \leftarrow \mathsf{Loop}(s_1, j_1, s_3, j_3) \cup \{T\}$;
  for $\alpha \to \beta \in \mathcal{R}$ **do**
    if $\alpha[i_\alpha] = \beta[i_\beta]$, $\alpha[j_\alpha] = \beta[j_\beta]$, *and* $\mathsf{type}(\beta) \in \mathsf{Loop}(s_1, i_\beta, s_2, j_\beta)$ **then**
      $\mathsf{Loop}(s_1, i_\alpha, s_2, j_\alpha) \leftarrow \mathsf{Loop}(s_1, i_\alpha, s_2, j_\alpha) \cup \{\mathsf{type}(\alpha)\}$;

Example 14. *Let the ruleset* $\mathcal{R}$ *consist of the following three linear rules:*

$$p(x, x, z) \to \exists w\, q(x, z, w) \qquad q(x, z, w) \to u(w, x) \qquad q(x, z, w) \to r(w, z)$$

and consider the types associated with the atoms appearing in these rules:

$$T_p = (p, \{\{1, 2\}, \{3\}\}) \quad T_q = (q, \{\{1\}, \{2\}, \{3\}\} \quad T_u = (u, \{\{1\}, \{2\}\}) \quad T_r = (r, \{\{1\}, \{2\}\})$$

For our query, we take the RPQ $r^- \cdot u^*(t_1, t_2)$ *where* $r, u \in \mathcal{P}_2$*. To apply Algorithm 1, we consider the corresponding NFA representation* $\mathbb{A}$ *with states* $\{s_0, s_f\}$ *(with* s_0 *initial and* s_f *final) and two transitions:* (s_0, r^-, s_f) *and* (s_f, u, s_f)*. We indicate which parts of the algorithm allow us to add type* T_p *to* $\mathsf{Loop}(s_0, 3, s_f, 1)$*:*

- *Type* T_r *is added to* $\mathsf{Loop}(s_0, 2, s_f, 1)$ *in Line 9 due to transition* (s_0, r^-, s_f)*.*
- *Type* T_u *is added to* $\mathsf{Loop}(s_f, 1, s_f, 2)$ *in Line 7 due to transition* (s_f, u, s_f)*.*
- *Type* T_q *is added to* $\mathsf{Loop}(s_0, 2, s_f, 3)$ *in Line 21 due to rule* $q(x, z, w) \to r(w, z)$*,* $T_r \in \mathsf{Loop}(s_0, 2, s_f, 1)$*, and the 2nd and 3rd arguments of* $q(x, z, w)$ *coinciding with the 2nd and 1st arguments of* $r(w, z)$*.*

- *Type T_q is added to* $\mathsf{Loop}(s_f, 3, s_f, 1)$ *in Line 21 due to rule* $q(x,z,w) \rightarrow u(w,x)$, $T_u \in \mathsf{Loop}(s_f, 1, s_f, 2)$, *and the 3rd and 1st arguments of* $q(x,z,w)$ *coinciding with the 1st and 2nd arguments of* $u(w,x)$.
- *Type T_q is added to* $\mathsf{Loop}(s_0, 2, s_f, 1)$ *in Line 18 because it is in both* $\mathsf{Loop}(s_0, 2, s_f, 3)$ *and* $\mathsf{Loop}(s_f, 3, s_f, 1)$.
- *Type T_p is added to* $\mathsf{Loop}(s_0, 3, s_f, 1)$ *in Line 21 due to rule* $p(x,x,z) \rightarrow \exists w\, q(x,z,w)$, $T_q \in \mathsf{Loop}(s_0, 2, s_f, 1)$, *and the 3rd and 1st arguments of* $p(x,x,z)$ *coinciding with the 2nd and 1st arguments of* $q(x,z,w)$. *Note that since the 1st and 2nd arguments of* $p(x,x,z)$ *are the same, the algorithm will also add T_p to* $\mathsf{Loop}(s_0, 3, s_f, 2)$.

The inclusion of type T_p in $\mathsf{Loop}(s_0, 3, s_f, 1)$ *allows one to infer for example that* $\{p(a,a,c)\}, \mathcal{R} \models r^- \cdot u^*(c,a)$.

The following propositions establish the correctness of Algorithm 1 and provide upper bounds on its running time. Note that the algorithm is independent from the data.

Proposition 15. *Let $\mathcal{R}$ be a set of linear rules, $\mathbb{A}$ be an NFA, T be a type whose predicate occurs in $\mathcal{R}$, and* Loop *be the table constructed by Algorithm 1 on input $(\mathcal{R}, \mathbb{A})$. Then $T \in \mathsf{Loop}(s, i, s', j)$ iff T admits an (s, i, s', j)-loop.*

Proof. Throughout the proof, we assume that for each type T, we have selected some atom α_T of type T.

($\Rightarrow$) We prove, by induction on the order of addition of types that whenever a type T is added to a cell in $\mathsf{Loop}(s, i, s', j)$, then T admits an (s, i, s', j)-loop. First, suppose that T is added to $\mathsf{Loop}(s, j, s, j')$ at Line 4. Then the trivial length-0 path $\alpha_T[j]$ (=$\alpha_T[j']$) in $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ witnesses that T admits an (s, j, s, j')-loop, since it is labeled by the empty word, which does not change the state of the automaton. Next, consider the case in which T is added to $\mathsf{Loop}(s_1, 1, s_2, 2)$ at Line 7. Then α_T takes the form $r(t_1, t_2)$, so $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ contains an r-labeled path from $\alpha_T[1]$ to $\alpha_T[2]$, which belongs to $\mathcal{L}_{\mathbb{A}}(s_1, s_2)$ since $s_2 \in \delta(s_1, r)$; hence, T admits an $(s_1, 1, s_2, 2)$-loop. The reasoning is similar for types added at Line 9 and Lines 12 to 15.

If T is added to $\mathsf{Loop}(s_1, j_1, s_3, j_3)$ at Line 18, then there must exist a state s_2 for which T has already been added to $\mathsf{Loop}(s_1, j_1, s_2, j_2)$ and $\mathsf{Loop}(s_2, j_2, s_3, j_3)$. By the induction assumption, there is a word u_1 (resp. u_2) in $\mathcal{L}_{\mathbb{A}}(s_1, s_2)$ (resp. $\mathcal{L}_{\mathbb{A}}(s_2, s_3)$) that labels a path in $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ from $\alpha_T[j_1]$ (resp. $\alpha_T[j_2]$) to $\alpha_T[j_2]$ (resp. $\alpha_T[j_3]$). By concatenating these paths, we obtain a path from $\alpha_T[j_1]$ to $\alpha_T[j_3]$ whose label u_1u_2 belongs to $\mathcal{L}_{\mathbb{A}}(s_1, s_3)$, which proves that T admits an (s_1, j_1, s_3, j_3)-loop.

Finally, assume that $\mathsf{type}(\alpha)$ is added to $\mathsf{Loop}(s_1, i_\alpha, s_2, j_\alpha)$ at Line 21 during the examination of rule $\alpha \rightarrow \beta \in \mathcal{R}$. Then it must be the case that $\alpha[i_\alpha] = \beta[i_\beta]$, $\alpha[j_\alpha] = \beta[j_\beta]$, and $\mathsf{type}(\beta) \in \mathsf{Loop}(s_1, i_\beta, s_2, j_\beta)$. By the induction assumption, $\mathsf{type}(\beta)$ admits an $(s_1, i_\beta, s_2, j_\beta)$-loop, which implies that there is a path p in $\mathsf{chase}(\beta, \mathcal{R})$ from $\beta[i_\beta]$ to $\beta[j_\beta]$ such that $\lambda(p) \in \mathcal{L}_{\mathbb{A}}(s_1, s_2)$. Since $\beta \in \mathsf{chase}(\{\alpha\}, \mathcal{R})$ we can apply Proposition 11 to infer that there is a path p' in $\mathsf{chase}(\{\alpha\}, \mathcal{R})$ from $\beta[i_\beta]$ to $\beta[j_\beta]$ such that $\lambda(p') \in \mathcal{L}_{\mathbb{A}}(s_1, s_2)$. As $\alpha[i_\alpha] = \beta[i_\beta]$, $\alpha[j_\alpha] = \beta[j_\beta]$, it follows that p' is also a path from $\alpha[i_\alpha]$ to $\alpha[j_\alpha]$, which witnesses that $\mathsf{type}(\alpha)$ admits an $(s_1, i_\alpha, s_2, j_\alpha)$-loop.

($\Leftarrow$) We suppose that T admits an (s, i, s', j)-loop and our aim is to show that T is added to $\mathsf{Loop}(s, i, s', j)$. We proceed by induction on the length of the shortest path in $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ that witnesses that T admits an (s, i, s', j)-loop.

Base case, path of length 0: In this case, there is length-0 path p in $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ from $\alpha_T[i]$ to $\alpha_T[j]$ that takes $\mathbb{A}$ from state s to state s'. We thus have $\lambda(p) = \varepsilon$, which implies that $s = s'$ and $i \sim_T j$. Hence T will be added to $\mathsf{Loop}(s, i, s', j)$ at Line 4.

Base case, path of length 1: The fact that T admits an (s, i, s', j)-loop is witnessed by a length-one path. Let $\alpha_T[i] q \alpha_T[j]$ be the witnessing path, where $q \in \mathcal{P}_2^{\pm}$ and $q \in \mathcal{L}_{\mathbb{A}}(s, s')$ (i.e., $s' \in \delta(s, q)$). There are three possibilities:

(a) if $q = r \in \mathcal{P}_2$ and $\alpha_T[i] \neq \alpha_T[j]$, then $\beta = r(\alpha_T[i], \alpha_T[j]) \in \mathsf{chase}(\{\alpha_T\}, \mathcal{R})$ and $\mathsf{type}(\beta)$ is added to $\mathsf{Loop}(s, 1, s', 2)$ in Line 7;

(b) if $q = r^-$ for $r \in \mathcal{P}_2$ and $\alpha_T[i] \neq \alpha_T[j]$, then $\beta = r(\alpha_T[j], \alpha_T[i]) \in \mathsf{chase}(\{\alpha_T\}, \mathcal{R})$, so $\mathsf{type}(\beta)$ is added to $\mathsf{Loop}(s, 2, s', 1)$ in Line 9;

(c) if $\alpha_T[i] = \alpha_T[j]$, then the atom

$$\beta = r(\alpha_T[i], \alpha_T[i]) = r(\alpha_T[j], \alpha_T[j]) = r(\alpha_T[i], \alpha_T[j]) = r(\alpha_T[j], \alpha_T[i])$$

belongs to $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$, so $\mathsf{type}(\beta)$ will be added to $\mathsf{Loop}(s, 1, s', 1)$, $\mathsf{Loop}(s, 1, s', 2)$, $\mathsf{Loop}(s, 2, s', 1)$ and $\mathsf{Loop}(s, 2, s', 2)$ in Lines 12-15.

In all three cases, the atom β belongs to $\mathsf{chase}(\{\alpha_T\}, \mathcal{R})$, which means that there exists a finite sequence of atoms $\alpha_T = \alpha_0, \ldots, \alpha_m = \beta$ such that α_{i+1} is generated by an application of $\rho_i \in \mathcal{R}$ mapping $\mathsf{body}(\rho_i)$ to α_i. We proceed by induction on the length m of the sequence.

First suppose that $m = 0$, which means that $\beta = \alpha_T$. In case (a), we have $T = \mathsf{type}(\beta)$, which means that $i = 1$, $j = 2$, and T is added to $\mathsf{Loop}(s, i, s', j)$ as desired. If we are in case (b), then we again have $T = \mathsf{type}(\beta)$ and T is added to $\mathsf{Loop}(s, 2, s', 1)$. Again in case (c), $\mathsf{type}(\beta) = T$ and T is added to $\mathsf{Loop}(s, 1, s', 1)$, $\mathsf{Loop}(s, 1, s', 2)$, $\mathsf{Loop}(s, 2, s', 1)$, and $\mathsf{Loop}(s, 2, s', 2)$.

Next suppose that $m > 1$ and that the property holds for all sequences of length at most $m-1$. As $\alpha_T[i]$ and $\alpha_T[j]$ appear in β, it holds that there exist i' and j' such that $\alpha_1[i'] = \alpha_T[i]$ and $\alpha_1[j'] = \alpha_T[j]$. As $\beta \in \mathsf{chase}(\{\alpha_1\}, \mathcal{R})$, $\mathsf{type}(\alpha_1)$ admits an (s, i', s', j')-loop, and there exists a derivation of length $m - 1$ that generates β from α_1. By our induction assumption for sequences of length at most $m - 1$, it holds that $\mathsf{type}(\alpha_1)$ has been added to $\mathsf{Loop}(s, i', s', j')$. As α_1 is generated from α_T by the application of a rule, Line 21 adds T to $\mathsf{Loop}(s, i, s', j)$.

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 $p = t_0 q_1 t_1 \ldots q_n t_n$. First consider the case in which t_k is contained in α_T for some $1 \leq k < n$, and let l be a position of t_k in α_T. There exists a path from t_0 to t_k of length strictly smaller than n, and similarly from t_k to t_n. By the induction assumption, $T = \mathsf{type}(\alpha_T)$ is in both $\mathsf{Loop}(s, i, s'', l)$ and $\mathsf{Loop}(s'', l, s', j)$ for some state s''. An application of Line 18 yields $T = \mathsf{type}(\alpha_T) \in \mathsf{Loop}(s, i, s', j)$. Next suppose there is no t_k $(1 \leq k < n)$ that occurs in α_T, which in particular means that $t_k \notin \{t_0, t_n\}$ for $1 \leq k < n$. Consider a finite derivation that generates all of the binary atoms from the path p. Let t_k be such that no other t_i $(1 \leq i < n)$ is created earlier in the derivation, and let β be the atom in which t_k is created. We prove that the path p is contained in $\mathsf{chase}(\{\beta\}, \mathcal{R})$. Indeed, the atoms $q_{k-1}(t_{k-1}, t_k)$ and $q_k(t_k, t_{k+1})$ must occur in $\mathsf{chase}(\{\beta\}, \mathcal{R})$ since β created t_k. By induction, we obtain that the same holds for all other atoms in p; this moreover implies that t_0 (resp. t_n) must occur in β, let us say at position i' (resp. j'). By the induction hypothesis, $\mathsf{type}(\beta)$ belongs to $\mathsf{Loop}(s, i', s'', k')$ and to $\mathsf{Loop}(s'', k', s', j')$ for some state s'', where k' is the position of t_k in β. Hence, by Line 18, $\mathsf{type}(\beta)$ is in the cell $\mathsf{Loop}(s, i', s', j')$. Following the same reasoning as in the base case for paths of length 1, by (repeated) application of Line 21, $\mathsf{type}(\alpha_T)$ is in the cell $\mathsf{Loop}(s, i, s', j)$, which concludes the proof. □

Proposition 16. *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 in the maximum predicate arity (hence polynomial for bounded-arity predicates). The first for loop runs in $O(n_t)$, the next two run in polynomial time, and the while loop is performed at most n_t times. □

There is a subtlety however that demands our attention, namely, the case of linear rulesets obtained via the single-head translation (see page 7). Indeed, the preceding result establishes polynomial time complexity for bounded-arity linear rules *with atomic heads*. However, if the original ruleset contains rules with multiple atoms in the head, then we must first apply the single-head translation, introducing new predicates which may have higher arities. This issue can be addressed by suitably modifying Algorithm 1 to take into account such new predicates. Specifically, for each such fresh predicate p_ρ (introduced for rule ρ), which occurs in the head of new rule $\mathsf{body}(\rho) \rightarrow \exists y_1 \ldots \exists y_\ell \, p_\rho(x_1, \ldots, x_k, y_1, \ldots, y_\ell)$, the modified algorithm only considers types for p_ρ that contain $\{\{k + 1\}, \ldots \{k + \ell\}\}$ (i.e. the positions that store existential variables are never merged with other

Algorithm 2: RPQ answering over linear rules

Input: An NFA $\mathbb{A} = (S, \Sigma, \delta, s_0, F)$, an instance I, a set of linear rules $\mathcal{R}$, $(a, b) \in \mathsf{terms}(I) \times \mathsf{terms}(I)$
Output: Yes if and only if (a, b) is a certain answer to the RPQ defined by $\mathbb{A}$

1 Compute Loop table for $\mathcal{R}$ and $\mathbb{A}$ (using Algorithm 1);
2 current = (a, s_0);
3 count = 0, max = $|S| \times |I|$;
4 **while** count < max *and* current $\notin \{(b, s_f) \mid s_f \in F\}$ **do**
5 Define (c, s) = current;
6 Guess (d, s') together with either $(s, r, s') \in \delta$ or T, i_c, i_d such that $T \in \mathsf{Loop}(s, i_c, s', i_d)$;
7 **if** (s, r, s') *was guessed and* $(I, \mathcal{R} \not\models r(c, d))$ **then**
8 **return** *No*
9 **else if** T, i_c, i_d *was guessed and* I *does not contain any atom* α *of type* T *such that* $\alpha[i_c] = c$ *and* $\alpha[i_d] = d$ **then**
10 **return** *No*
11 **else**
12 current = (d, s'), count = count +1;
13 **if** current= (b, s_f) *for some* $s_f \in F$ **then return** *Yes* **else return** *No*;

positions). Note that for the original predicates, the modified algorithm considers all possible types (just like in Algorithm 1). The following result shows that the modified algorithm satisfies the required properties.

Proposition 17. *The modified Algorithm 1 runs in polynomial time for rulesets obtained by applying the single-head translation to linear rulesets with bounded predicate arity. Moreover, for any linear ruleset $\mathcal{R}$, whose single-head translation is $\mathcal{R}'$, any NFA $\mathbb{A}$, and any type T based upon a predicate in $\mathcal{R}$, type T belongs to $\mathsf{Loop}'(s, i, s', j)$ in the table produced by running the modified algorithm on input $(\mathcal{R}', \mathbb{A})$ iff T admits an (s, i, s', j)-loop (w.r.t. $\mathcal{R}$).*

Proof. The only predicates in $\mathcal{R}'$ whose arity exceeds the maximum predicate arity of $\mathcal{R}$ are the newly introduced predicates, and each such predicate p_ρ can only appear in the head of a single rule of the form $\mathsf{body}(\rho) \rightarrow \exists y_1 \ldots \exists y_\ell\, p_\rho(x_1, \ldots, x_k, y_1, \ldots, y_\ell)$ and in the body of rules $p_\rho(x_1, \ldots, x_k, y_1, \ldots, y_\ell) \rightarrow q(\vec{u})$ (for each $q(\vec{u})$ which is a head atom of ρ). As a consequence, the only types with predicate p_ρ which are necessary to consider in Algorithm 1 are those which contain the trivial partition $\{\{k+1\}, \ldots \{k+\ell\}\}$. Indeed, by using the same arguments as in Prop. 16, we can show that if T is any type for an original predicate in $\mathcal{R}$, then T belongs to $\mathsf{Loop}'(s, i, s', j)$ in the table produced by the modified algorithm on input $(\mathcal{R}', \mathbb{A})$ iff T admits an (s, i, s', j)-loop. Finally, to establish polynomial complexity, it suffices to remark that there can only be polynomially many types for the new predicates (since k is bounded by a constant). □

The remainder of the decision procedure relies on the following ideas (Algorithm 2): starting from a constant a and the initial state of $\mathbb{A}$, we guess the next constant d in I on a path from a to b and the state of $\mathbb{A}$ after taking this step (Line 6). Note that a and d may be equal. We then check that this choice is valid, i.e., there is indeed a path from a to the guessed constant d which takes the automaton from the initial state to the current guessed state. This can be done either by checking that a corresponding binary atom is entailed (Lines 6-7), or by checking that a path going through the anonymous part of the chase allows us to reach the next constant in the required state, using the Loop table (Lines 8-9). We repeat this procedure until we reach the constant b in a final state, or hit the maximal path length.

The following lemma proves an invariant that will be used to establish the correctness of Algorithm 2.

Lemma 18. *At the beginning of each iteration of the while loop of Algorithm 2, it holds that there is a path in* $\mathsf{chase}(I, \mathcal{R})$ *from a to the first element of* `current` *that takes the NFA $\mathbb{A}$ from the initial state s_0 to the state in the second argument of* `current`.

Proof. At the beginning of the first iteration of the while loop, `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 `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, r, s') or (T, i_c, i_d) has been guessed, and the corresponding check was successful. Let us consider each case:

- if (s, r, s') has been guessed and checked, then $r \in \mathcal{P}_2^{\pm}$, and there is a path from a_i to a_{i+1} in $\mathsf{chase}(I, \mathcal{R})$ labeled by r. Moreover, r labels an edge from s to s' in $\mathbb{A}$. We can thus define $w_{i+1} = w_i.r$
- if (T, i_c, i_d) has been guessed, it means that T belongs to $\mathtt{Loop}(s_i, i_c, s_{i+1}, i_d)$. By the definition of `Loop`, there is a path p (in the anonymous 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 9. As $I, \mathcal{R} \models \alpha$, where $\mathsf{type}(\alpha) = T$, 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 $w_{i+1} = w_i.p$. □

The next proposition establishes the correctness of Algorithm 2. To prove the correctness when the algorithm outputs Yes, we rely on the invariant given by the previous lemma, which allows us to construct a witnessing path in $\mathsf{chase}(I, \mathcal{R})$. To prove the correctness when the algorithm outputs No, we consider any path of minimal length from a to b that satisfies the input RPQ and build a sequence of guesses associated with it that is necessarily accepted by the algorithm.

Proposition 19. *There is an execution of Algorithm 2 that outputs Yes if and only if 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 Lemma 18, 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 shows 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 $\mathsf{chase}(I, \mathcal{R})$ 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 $\mathsf{terms}(I)$;
- for any i, $s_i = s'_{k_i}$.

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

- α is such that $\alpha \in I$ and $\mathsf{type}(\alpha) \in \mathtt{Loop}(s_i, i_c, s_{i+1}, i_d)$;
- 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, we can 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 $(a_i, s_i, \mathsf{aux}_i)$ 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 7 and 9 fails. Let $(a_i, s_i, \mathsf{aux}_i)$ 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 $\mathsf{chase}(I, \mathcal{R})$. Thus, $r_{i+1}(a_{k_i}, a_{k_{i+1}})$ is entailed by I and $\mathcal{R}$, and the check at Line 7 is successful. If aux_i is of the form $(\mathsf{type}(\alpha), i_c, i_d)$, then there is $\alpha \in I$ such that $\mathsf{type}(\alpha) \in \mathsf{Loop}(s_i, i_c, s_{i+1}, i_d)$, 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 Line 9. Thus the defined sequence never triggers a rejection from Algorithm 2, which concludes the proof. □

By analyzing the time and space required by Algorithm 2, we obtain the following upper-bounds for RPQ Answering in the presence of linear existential rules.

Theorem 20. *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: checking entailment of an atom with linear rules and accessing the contents of the Loop table (more precisely, deciding whether $T \in \mathsf{Loop}(s, i_c, s', i_d)$). 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 (more precisely, AC_0, see Section 2.4), 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 (Section 2.4), while Loop can be computed in polynomial time (even if multiple head atoms are allowed, due to Proposition 17). The overall algorithm is thus in PTime with bounded arity. In the unbounded arity case, the entailment checks can be performed in PSpace (Section 2.4), while the Loop table can be computed in ExpTime: the overall algorithm thus runs in ExpTime. □

4 RPQ Answering under Linear Rules: Lower Bound

The data complexity (resp. combined complexity) of RPQs under linear rules (resp. linear rules with bounded arity) is already known to be NL-hard (resp. PTime-hard) (Bienvenu et al. 2015). As these bounds match the upper bounds obtained in the preceding section, we focus on providing a matching ExpTime lower bound for the combined complexity of evaluating RPQs under linear rules of unbounded arity.

We recall that ExpTime can be reformulated as APSpace, the class of problems that can be solved in polynomial space by an alternating Turing Machine (ATM). Our proof relies on this equivalence and shows that a PSpace ATM can be simulated by means of linear rules. Without loss of generality, we consider an ATM where each non-final universal (resp. existential) state has exactly two existential (resp. universal) sucessors.

It is already known that PSpace TMs can be simulated by means of linear rules (Gottlob and Papadimitriou 2003). In the following, we explain how to adapt this construction to simulate ATMs. To simplify the presentation, we will use rules with both constants and multiple atoms in the head. This can be done without loss of generality, since we can apply classical transformations to turn such rules into constant-free atomic-head rules (see, e.g., Definition 1.37 in (Rocher 2016) for the removal of constants, and Section 2.1 for the single-head translation). These translations are both polynomial in the size of the input ruleset and preserve the property of being a linear ruleset.

The key point in the construction from (Gottlob and Papadimitriou 2003) is the following: the configuration of a PSpace TM $\mathcal{M}$ is represented 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 that encodes 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 an ATM: to each atom, we add two positions, that will act as "input" and "output" positions. Moreover, we maintain the following property: for any atom α encoding a configuration c, there is a path, whose edges are all labeled by the same predicate p, from the term i_c in input position of α to the term o_c in output position of α, entailed by chase$(I_{\{\alpha\}}, \mathcal{R}_\mathcal{M})$ if and only if the configuration represented by α is accepted by $\mathcal{M}$. Acceptance occurs 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 (Gottlob and Papadimitriou 2003). Given an ATM $\mathcal{M}$ and an input x, such that $\mathcal{M}$ runs in $P(|x|)$ space for a polynomial P, 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 $\mathcal{M}$ and the current state of $\mathcal{M}$. Adding input and output positions, this can be encoded by a predicate conf of arity $2P(|x|) + 3$:

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

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 $\mathcal{M}$ 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 accepting state q_f, we create the following rule (where q_f denotes a constant):

$$\mathsf{conf}(i_c, q_f, \ldots, o_c) \rightarrow p(i_c, o_c) \tag{1}$$

For each existential state q and each transition $\delta(q, \gamma) = \{(q', \gamma', L), (q'', \gamma'', L)\}$, and for every position i on the tape, we create the following rule (where q, q', q'', γ, γ' and γ'' denote constants):

$$\begin{aligned}\mathsf{conf}(i_c, q, cell_1, cur_1, \ldots, cell_{i-1}, 0, \gamma, 1, \ldots, o_c) \rightarrow \\ \exists i_{c'}, o_{c'}, i_{c''}, o_{c''}\ \mathsf{conf}(i_{c'}, q', cell_1, cur_1, \ldots, cell_{i-1}, 1, \gamma', 0, \ldots, o_{c'}), \\ \mathsf{conf}(i_{c''}, q'', cell_1, cur_1, \ldots, cell_{i-1}, 1, \gamma'', 0, \ldots, o_{c''}), \\ p(i_c, i_{c'}), p(o_{c'}, o_c), p(i_c, i_{c''}), p(o_{c''}, o_c)\end{aligned} \tag{2}$$

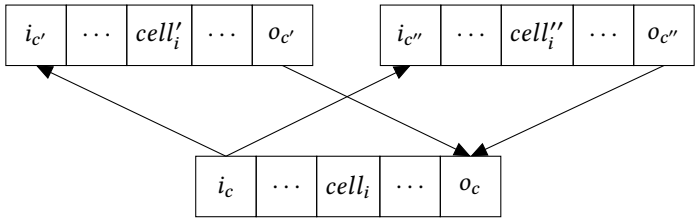

Fig. 1. The existential gadget

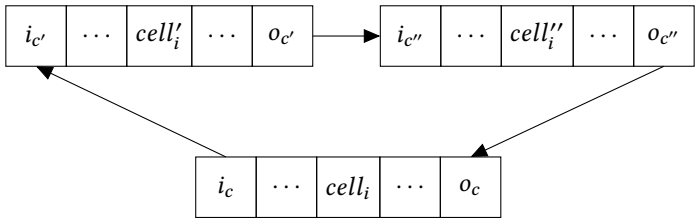

Fig. 2. The universal gadget

For each universal state q and each transition $\delta(q, \gamma) = \{(q', \gamma', L), (q'', \gamma'', L)\}$, and for every position i on the tape, we create the following rule (with the same constants as in the preceding rule):

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

We proceed similarly to translate transitions in which the head is moving to the right. Figure 1 (resp. Figure 2) illustrates the functioning of rules of type (2) (resp. type (3)). We denote by $\mathcal{R}_{\mathcal{M},x}$ the set of all the rules defined above. Note that x is required to determine the arity of conf.

The following proposition establishes the correctness of the reduction.

Proposition 21. *Let $\mathcal{M}$ be a PSpace ATM and let α be an atom of* $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$ *representing a configuration $c(\alpha)$. Then $c(\alpha)$ is an accepting configuration of $\mathcal{M}$ if and only if there is a path in* $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$ *from $i(\alpha)$ to $o(\alpha)$ whose label belongs to p^*.*

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

- If C_α contains a single atom, then there can be no path in $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$ 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, with $n > 2$. 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 that 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 $o(\alpha)$, since 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 a 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. 2). 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 $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$.
- 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 $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. 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 $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. 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 $\mathsf{chase}(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. □

Now let $\mathcal{M}$ be a PSpace ATM, x be an input to $\mathcal{M}$, and α be the unique atom in $I_{\mathcal{M},x}$. Then by Proposition 21, $c(\alpha)$ is an accepting configuration of $\mathcal{M}$ if and only if $I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x} \models p^*(i(\alpha), o(\alpha))$; in other words: if and only if $(i(\alpha), o(\alpha))$ is a certain answer to the RPQ $p^*(x, y)$ on the KB $(I_{\mathcal{M},x}, \mathcal{R}_{\mathcal{M},x})$. This, together with known results, yields the following lower bounds:

Theorem 22. *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. The preceding reduction can be used to show that atomic query entailment under rulesets composed of linear rules and transitivity rules (AQELT) is ExpTime-hard in combined complexity, already with a single transitivity rule. Indeed, a pair of constants (a, b) is an answer to an RPQ $p^+(x, y)$ on a KB $\mathcal{K} = (I, \mathcal{R})$ with $\mathcal{R}$ a linear ruleset iff the atomic Boolean query $p'(a, b)$ is entailed by the KB $(I, \mathcal{R} \cup \{p(x, y) \rightarrow p'(x, y); p'(x, y) \wedge p'(y, z) \rightarrow p'(x, z)\}$, where p' is a fresh predicate introduced to avoid interactions between the new rules and those from $\mathcal{R}$.

Assuming ExpTime≠PSpace, our result is in contradiction with Theorem 5 in (Baget et al. 2015a), which purports to show a PSpace upper bound for the AQELT problem. 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 their procedure runs in exponential time, rather than in polynomial space (their NL upper bound in data complexity remains valid). Combining our lower bound with

their procedure shows that the AQELT problem is ExpTime-complete in combined complexity. The incorrect claim from (Baget et al. 2015a) was corrected in the companion report with full proofs (Baget et al. 2015b).

5 CRPQ Answering under Linear Rules: Upper Bound

We now study the complexity of CRPQ answering under linear rules. The algorithm we propose to solve this problem is based on two notions. First, the notion of a *proof scheme*, which (under some validity conditions) captures a finite portion of the chase. Second, the notion of a *match* of a (Boolean) CRPQ in a proof scheme, which attests that this CRPQ is entailed by the portion of the chase captured by this proof scheme. The algorithm roughly works as follows: it enumerates all proof schemes of 'small' size and checks whether one of them is valid and admits a match of the query. To obtain our complexity results, we will upper bound both the size of the proof schemes that need to be considered and the cost of the validity check.

Section 5.1 introduces the fundamental notions. It concludes with Proposition 29, which states that a query q is entailed by a knowledge base if and only if there exists a match of q in some valid proof scheme. Section 5.2 proves (the hard direction of) that proposition, by exhibiting a suitable proof scheme whenever q is entailed by the knowledge base. Finally, Section 5.3 analyzes the computational resources required to enumerate proof schemes and to check their validity as well as the existence of a match of q in one of them.

5.1 Proof Schemes

We recall that a CRPQ q may contain both standard atoms and path atoms whose predicate is defined by an automaton. Given a pair of atoms, it will be important to keep track of the paths in the chase that link terms occurring in these atoms and correspond to words that can be read by the automata appearing in q, i.e., that take one of these automata from a state s_1 to a state s_2. The next definition of *transitions* between atoms plays this role.

Definition 23 (Transition). *Let A be a set of atoms. Let α_1 and α_2 be two atoms of A of arity k_1 and k_2 respectively. Given a query q, the* set of transitions from α_1 to α_2 in A, *denoted by $\mathcal{T}_{q,A}(\alpha_1, \alpha_2)$, is the set of quadruples (i_1, s_1, i_2, s_2) with $1 \leq i_1 \leq k_1$ and $1 \leq i_2 \leq k_2$, such that there are an automaton $\mathbb{A}$ in q and a path p in A going from $\alpha_1[i_1]$ to $\alpha_2[i_2]$ such that $\lambda(p) \in \mathcal{L}_{\mathbb{A}}(s_1, s_2)$. More generally, by* transition*, we will mean a quadruple (i_1, s_1, i_2, s_2) where s_1, s_2 are states of an automaton in q, and i_1, i_2 do not exceed the maximum predicate arity in $I \cup \mathcal{R}$.*

This allows us to introduce the key technical notion underlying our approach, namely, *proof schemes*. Briefly, a proof scheme is a directed forest whose nodes are atoms, together with transitions for different pairs of atoms in this forest. Intuitively, under some validity conditions defined later (Definition 27), a proof scheme corresponds to a subset of the chase, with the transitions placing requirements on the paths linking these atoms.

Definition 24 (Proof scheme). *A* proof scheme *is a pair $\mathbb{P} = (\mathcal{F}, \mathcal{T})$ such that:*

(1) $\mathcal{F} = (V, E)$ is a directed forest;
(2) V is a set of atoms;
(3) for each null $\bot$, the set of nodes in which $\bot$ occurs form a connected subgraph of $\mathcal{F}$;
(4) $\mathcal{T}$ is a set of pairs $(\tau, (\alpha_1, \alpha_2))$, where α_1 and α_2 belong to V and τ is a transition from α_1 to α_2.
(5) for each $(\tau, (\alpha_1, \alpha_2)) \in \mathcal{T}$, either $\alpha_1 = \alpha_2$, $(\alpha_1, \alpha_2) \in E$, $(\alpha_2, \alpha_1) \in E$, or both α_1 and α_2 are roots.

We denote by $\mathsf{vars}(\mathbb{P})$ *and* $\mathsf{terms}(\mathbb{P})$ *the variables and terms occurring in atoms of V, i.e.,* $\mathsf{vars}(\mathbb{P}) = \mathsf{vars}(V)$ *and* $\mathsf{terms}(\mathbb{P}) = \mathsf{terms}(V)$.

In a proof scheme, transitions are assigned to pairs of atoms that comply with specific conditions: the atoms are equal, or both are roots, or one is the parent of the other in a tree. To check the existence of paths that link terms occurring in arbitrary pairs of atoms, we can compose these transitions. The *saturation* of a proof scheme, defined next, enriches the proof scheme with all the transitions that can be obtained by composition.

Definition 25 (Saturation of a proof scheme). *Let $\mathbb{P} = (\mathcal{F}, \mathcal{T})$ be a proof scheme. The* saturation *of $\mathbb{P}$ is the pair $(\mathcal{F}, \mathcal{T}')$ where $\mathcal{T}'$ is the smallest set such that $\mathcal{T} \subseteq \mathcal{T}'$ and $\{((i_1, s_1, i_2, s_2), (\alpha_1, \alpha_2)), ((i_2, s_2, i_3, s_3), (\alpha_2, \alpha_3))\} \subseteq \mathcal{T}'$ implies $((i_1, s_1, i_3, s_3), (\alpha_1, \alpha_3)) \in \mathcal{T}'$.*

Definition 26 (Match in a proof scheme). *Let $\mathbb{P} = (\mathcal{F}, \mathcal{T})$ be a proof scheme with $\mathcal{F} = (V, E)$. A* match *of q in $\mathbb{P}$ is a mapping π from $\mathsf{vars}(q)$ to $\mathsf{terms}(\mathbb{P})$ such that:*

- *if α is a standard atom in q, then $\pi(\alpha) \in V$;*
- *if $\alpha = \Lambda(x, y)$ is a path atom in q with associated NFA $\mathbb{A}$, then there exists a pair $((i_x, s_x, i_y, s_y), (\alpha_x, \alpha_y))$ in the saturation of $\mathbb{P}$ such that $\pi(x) = \alpha_x[i_x]$, $\pi(y) = \alpha_y[i_y]$ and s_x (resp. s_y) is the initial (resp. a final) state of $\mathbb{A}$.*

Note that, in the second point of the previous definition, x and y are not necessarily variables and we silently extend π to constants by setting $\pi(c) = c$ for any constant c. The problem of deciding if there is a match of a given query q in a given proof scheme $\mathbb{P}$ is NP-complete. Indeed, checking whether a candidate mapping is indeed a match of q in $\mathbb{P}$ can be done in polynomial time, since the saturation of a proof scheme can be computed in polynomial time; NP-hardness can be shown by reducing the homomorphism problem (given an instance I and a CQ q, is there a homomorphism from q to I?), with I being translated into a trivial proof scheme where $V = I$, $E = \emptyset$ and $\mathcal{T} = \emptyset$.

As already mentioned, not all proof schemes actually encode a part of the chase. More precisely, a proof scheme is said to be *valid* if it can be embedded into the chase by a mapping called a 'witnessing embedding'. Then, given a valid proof scheme $\mathbb{P}$ and a query q, the composition of a match of q in $\mathbb{P}$ and a witnessing embedding of $\mathbb{P}$ yields a match of q in the chase. We will also show that, whenever there is a match of q in the chase, there is a match of q in some valid proof scheme.

Definition 27 (Valid proof scheme and witnessing embedding). *A proof scheme $\mathbb{P} = (\mathcal{F}, \mathcal{T})$ with $\mathcal{F} = (V, E)$ is* valid *w.r.t. $(I, \mathcal{R}, q)$ if there is a mapping ψ from $\mathsf{vars}(\mathbb{P})$ to $\mathsf{terms}(\mathsf{chase}(I, \mathcal{R}))$ such that:*

(1) for each root atom $\alpha \in V$, $\psi(\alpha) = \alpha$ and $\alpha \in I$;
(2) for each atom $\alpha \in V$, $\psi(\alpha) \in \mathsf{chase}(I, \mathcal{R})$;
(3) for each edge $(\alpha_1, \alpha_2) \in E$, there is an $\mathcal{R}$-derivation from $\{\psi(\alpha_1)\}$ to (an extended instance that contains) $\psi(\alpha_2)$.
(4) for each $(\tau, (\alpha_1, \alpha_2)) \in \mathcal{T} : \tau \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_1), \psi(\alpha_2))$.

We say ψ is a witnessing embedding *of $\mathbb{P}$.*

Note that the saturation of a proof scheme is generally not a proof scheme, as the added transitions may label arbitrary pairs of atoms. However, and this is a point worth noting even if it is immediate, the saturation of a valid proof scheme also fulfills all the validity conditions from Definition 27.

Example 28 (Running Example—continued). *Let $\mathcal{R}$ be the set of linear rules from Example 1:*

(ρ_1) $\mathsf{isFriendOf}(x, y) \rightarrow \mathsf{isFriendOf}(y, x)$
(ρ_2) $\mathsf{isFriendOf}(x, y) \rightarrow \mathsf{follows}(x, y)$
(ρ_3) $\mathsf{follows}(x, y) \rightarrow \exists m\, \mathsf{message}(m, x, y)$

Consider again the instance $I = \{\mathsf{follows}(\mathsf{B}, \mathsf{A}), \mathsf{isFriendOf}(\mathsf{C}, \mathsf{A}), \mathsf{isFriendOf}(\mathsf{C}, \mathsf{B})\}$ and the following CRPQ:

$$q_6() = \exists y, m.\ \mathsf{follows}.\mathsf{follows}^*(A, y) \wedge \mathit{message}(m, y, A)$$

We assign to the path predicate $\mathsf{follows}.\mathsf{follows}^*$ the NFA representation $\mathbb{A}$ with states $\{s_0, s_f\}$ and two transitions: $(s_0, \mathsf{follows}, s_f)$, $(s_f, \mathsf{follows}, s_f)$. Recall that query q_6 can be matched to the following part of $\mathsf{chase}(I, \mathcal{R})$: $\{\mathsf{follows}(A, C), \mathsf{follows}(C, B), \mathsf{message}(m_0, B, A)\}$.

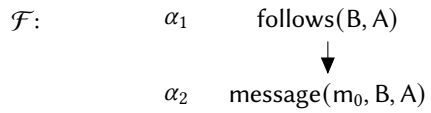

Fig. 3. Valid proof scheme $(\mathcal{F}, \mathcal{T})$, with $\mathcal{T} = \{((2, s_0, 1, s_f), (\alpha_1, \alpha_1))\}$ (Example 28)

A proof scheme $\mathbb{P} = (\mathcal{F}, \mathcal{T})$ for q_6 is pictured in Figure 3. The forest $\mathcal{F}$ contains a single tree, having two nodes. The only appearing null is m_0 and it occurs in a single node, which trivially forms a connected subgraph of $\mathcal{F}$. The set of transitions $\mathcal{T}$ contains a single transition $(\tau, (\alpha_1, \alpha_1))$, where $\tau = (2, s_0, 1, s_f)$. Proof scheme $\mathbb{P}$ is valid w.r.t. $(I, \mathcal{R}, q_6)$, as attested by a witnessing embedding $\psi = \{m_0 \mapsto m_0\}$. Note that the transition holds due to the atoms follows(A, C) *and* follows(C, B) *in I. Query q_6 has a match in $\mathbb{P}$ given by $\pi = \{y \mapsto B, m \mapsto m_0\}$.*

The next proposition states that CRPQ answering can be recast as deciding whether there is a match of the query in a 'small' valid proof scheme.

Proposition 29. *$I, \mathcal{R} \models q$ iff there exists a match of q in some valid proof scheme of polynomial size in q.*

Proof. ($\Leftarrow$) Let $\mathbb{P}$ be a valid proof scheme and ψ be a witnessing embedding of $\mathbb{P}$ in chase$(I, \mathcal{R})$. Let π be a match of q in $\mathbb{P}$. We claim that $\psi \circ \pi$ is a match of q in chase$(I, \mathcal{R})$, which proves that $I, \mathcal{R} \models q$. Indeed, take any atom α of q. If α is a standard atom, $\pi(\alpha) \in V$, hence $\psi \circ \pi(\alpha) \in$ chase$(I, \mathcal{R})$. If α is a path atom, let $\alpha = \Lambda(x, y)$, then there is a pair $((i_x, s_x, i_y, s_y), (\alpha_x, \alpha_y))$ in the saturation of $\mathbb{P}$, such that $\pi(x) = \alpha_x[i_x]$ and $\pi(y) = \alpha_y[i_y]$, with s_x and s_y initial and final states in the automaton associated with Λ. As $\mathbb{P}$ is valid, its saturation also fulfills the validity conditions. Hence, there is a path p from $\psi(\pi(x))$ to $\psi(\pi(y))$ in chase$(I, \mathcal{R})$ such that $\lambda(p)$ belongs to $\mathcal{L}(\Lambda)$. We conclude that $\psi \circ \pi$ is a match of q in chase$(I, \mathcal{R})$.
($\Rightarrow$) This part of the proof is addressed in Section 5.2. □

5.2 Proof of Proposition 29

We now prove that if $I, \mathcal{R} \models q$, then one can build a valid proof scheme $\mathbb{P}$ of polynomial size in q, such that there is a match of q in $\mathbb{P}$. For that, we rely on a classical tool, namely the *chase graph* associated with a derivation, which keeps track of how atoms are generated in this derivation. Nodes are labeled by the atoms occurring in the derivation and there is an edge (v, v') if a trigger in the derivation is applied on the atom that labels v and leads to generate the atom that labels v'. Note that we only keep track of rule applications that produce new atoms and that the derivation may not be fair (yet).

Definition 30 (Chase graph associated with a derivation). *Let I be an instance and $\mathcal{R}$ be a set of linear rules. The* chase graph *associated with a (possibly infinite) $\mathcal{R}$-derivation D from I, denoted by $\mathcal{CG}_D$, is a (possibly infinite) node-labeled directed forest built as follows (we use ℓ for the labeling function):*

- *its set of nodes is in bijection with* atoms(D) *via ℓ;*
- *for each trigger (ρ_i, π_i) in D that* directly generates *an atom α_i, there is an edge (v, v') where $\ell(v) = \pi_i(\text{body}(\rho_i))$ and $\ell(v') = \alpha_i$.*

The set of atoms that label the nodes of $\mathcal{CG}_D$ is denoted by atoms$(\mathcal{CG}_D)$.

Note that $\mathcal{CG}_D$ is indeed a forest because each atom outside I is directly generated exactly once in D and there is a single node labeled by this atom. Also note that atoms$(\mathcal{CG}_D) =$ atoms(D).

The next example will be used to illustrate the technical notions of this section, beginning with that of a chase graph.

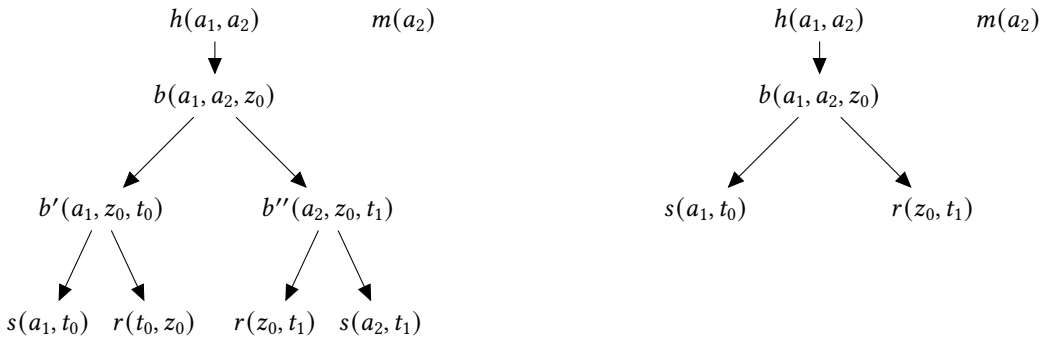

Fig. 4. Chase graph $\mathcal{CG}_D$ on the left and a backbone of π on the right (Example 31)

Example 31. *Let $I = \{h(a_1, a_2), m(a_2)\}$ and $\mathcal{R}$ containing the following rules:*

$R_1 : h(x, y) \rightarrow \exists z\, b(x, y, z)$
$R_2 : b(x, y, z) \rightarrow \exists t\, b'(x, z, t)$ $\quad$ $R_5 : b(x, y, z) \rightarrow \exists t\, b''(y, z, t)$
$R_3 : b'(x, y, z) \rightarrow s(x, z)$ $\quad$ $R_6 : b''(x, y, z) \rightarrow r(y, z)$
$R_4 : b'(x, y, z) \rightarrow r(z, y)$ $\quad$ $R_7 : b''(x, y, z) \rightarrow s(x, z)$

The chase graph associated with a (fair) $\mathcal{R}$-derivation D from I to I' is pictured in Figure 4, left. Let $q = \exists x_1 \exists x_2 \exists x_3 \exists x_4\ h(x_1, x_2) \wedge m(x_2) \wedge s^*(x_1, x_3) \wedge r^*(x_3, x_4) \wedge s^*(x_2, x_4)$. There is a match π of q in I' defined by: $\{x_1 \rightarrow a_1, x_2 \rightarrow a_2, x_3 \rightarrow t_0, x_4 \rightarrow t_1\}$.

To simplify writing, when a term t occurs in an atom $\alpha = \ell(v)$, we will say that α contains t, and by extension that v *contains* t. This is extended to a set of nodes and a subgraph. A chase graph $\mathcal{CG}_D$ can also be seen as a tree decomposition of the set of atoms $atoms(\mathcal{CG}_D)$, in the following sense: $\mathcal{CG}_D$ is a forest structure such that *(i)* nodes are labeled by atoms from $atoms(\mathcal{CG}_D)$, *(ii)* any atom of $atoms(\mathcal{CG}_D)$ is the label of some node of $\mathcal{CG}_D$ and *(iii)* for any null $\bot$, the set of nodes that contain $\bot$ forms a connected (rooted) subgraph of $\mathcal{CG}_D$; for any constant c, the set of nodes *in a tree that contains* c form a connected (rooted) subgraph of $\mathcal{CG}_D$. Next, the item *(iii)* is called the *connectivity property* of the chase graph.

The next proposition relates paths in a chase graph, which correspond to (undirected) paths of atoms, and paths in the chase, which are (directed) paths of terms. Informally, it says that, given any atoms α_x and α_y in the same tree of the chase graph, any path of terms in the chase, from a term x occurring in α_x to a term y occurring in α_y, goes through any atom along the unique shortest path from α_x to α_y in the chase graph.

Proposition 32 (Path property). *Let $\mathcal{CG}_D$ be a chase graph associated with a derivation D. Let v_x and v_y be nodes in the same tree of $\mathcal{CG}_D$, such that v_x contains term x and v_y contains term y. Let v be any node on the unique shortest (undirected) path from v_x to v_y in $\mathcal{CG}_D$. Then, any path (of terms) p from x to y in $atoms(\mathcal{CG}_D)$ can be written $p = p_1.p_2$, where p_1 and p_2 are paths (of terms) in* $\mathsf{chase}(I, \mathcal{R})$*, respectively from x to z and from z to y, such that v contains z.*

Proof. Let $p = t_0 r_1 t_1 \dots r_n t_n$ such that $t_0 = x$ and $t_n = y$. If v contains x (resp. y), the statement trivially holds: we take p_1 empty and $p_2 = p$ (resp. $p_1 = p$ and p_2 empty). Otherwise (we know that $v \neq v_x$ and $v \neq v_y$), removing v from $\mathcal{CG}_D$ splits the graph into several connected components, including one, that we call C_x (resp. C_y) containing v_x (resp. v_y). Let us consider the smallest i in p such that t_i occurs in a connected component $C \neq C_x$. There must be such an i, since $t_n = y$ occurs in C_y. We show that t_i occurs in v, i.e., t_i is a term z as

required by the property. Since $r_i(t_{i-1}, t_i) \in atoms(\mathcal{CG}_D)$, there is a node v' of $\mathcal{CG}_D$ labeled by $r_i(t_{i-1}, t_i)$. By hypothesis on t_i, t_{i-1} does not appear outside $C_x \cup \{v\}$, hence v' belongs to $C_x \cup \{v\}$. Either $v' = v$ (and t_i appears in v), or v' belongs to C_x: in this case, t_i appears both in C_x and in $C \neq C_x$, hence, by the connectivity property on t_i, t_i must also appear in v, which concludes the proof. □

Example 31 (continued). See also Figure 4 (left). Let $p = a_1\ s\ t_0\ r\ z_0\ r\ t_1\ s^-\ a_2$ be a path of terms, which occurs in the chase. We can see that p has no direct connection with the structure of the chase graph pictured in Figure 4 (left). Now, take, e.g., the nodes v_{a_1} labeled by $b'(a_1, z_0, t_0)$ and v_{a_2} labeled by $s(a_2, t_1)$, and consider the shortest path of atoms from v_{a_1} to v_{a_2} in the chase graph, which goes through $b(a_1, a_2, z_0)$, and $b''(a_2, z_0, t_1)$. W.r.t. the atom $b(a_1, a_2, z_0)$, p can be written $p_1.p_2$ with $p_1 = a_1\ s\ t_0\ r\ z_0$ and $p_2 = z_0\ r\ t_1\ s^-\ a_2$. W.r.t. the atom $b''(a_2, z_0, t_1)$, p can be decomposed in the same way, or also $p_1''.p_2''$ with $p_1'' = a_1\ s\ t_0\ r\ z_0\ r\ t_1$ and $p_2'' = t_1\ s^-\ a_2$.

When $I, \mathcal{R} \models q$, there is a finite derivation D from I to an extended instance I_n with a match π of q in I_n. We will now build a valid proof scheme from D and π. To do so, guided by π, we first select some atoms in I_n: *(i)* the images of the standard atoms of q, *(ii)* for each variable of q mapped to a null, an atom containing its image, and *(iii)* some atoms that allow to check for the existence of required paths. Note that some terms of q may appear only in path atoms, which justifies *(ii)*. Each of these atoms being the label of a unique node in $\mathcal{CG}_D$, the selected set of atoms is naturally structured in a forest induced by $\mathcal{CG}_D$. From this 'backbone', we then build the intended proof scheme by adding all the transitions that appear between eligible pairs of atoms.

Next, we use the notation $v_1 \prec v_2$ to denote that v_1 is a *strict ancestor* of v_2 in the considered chase graph. Also, for v and v' in the same tree of a chase graph, $\text{glb}(v, v')$ denotes their *greatest lower bound* (i.e., their deepest common ancestor) in this graph.

Definition 33 (Backbone of a match). *Let D be an $\mathcal{R}$-derivation from I to I_n, such that $I_n \models q$. Let π be a match of q in I_n. A* backbone *of π is a directed forest (V, E) defined as follows:*

- *V is a subset of I_n restricted to the following atoms:*
  - *for each standard atom $\alpha \in q$, the atom $\pi(\alpha)$;*
  - *for each variable x of q such that $\pi(x)$ is a null, some $\alpha_x \in I_n$ that contains $\pi(x)$;*
  - *for each atom $\alpha \in V$, the atom $\ell(v_r)$, where v_r is the root of the tree of $\mathcal{CG}_D$ in which the node v with $\ell(v) = \alpha$ occurs;*
  - *for each pair $(\alpha_1, \alpha_2) \in V \times V$ such that $\alpha_1 = \ell(v_1), \alpha_2 = \ell(v_2)$, and v_1, v_2 belong to the same tree of $\mathcal{CG}_D$, the atom $\ell(v)$, where $v = \text{glb}(v_1, v_2)$ in $\mathcal{CG}_D$;*
- *$E \subseteq V \times V$ is the set of edges induced by $\mathcal{CG}_D$, i.e., it contains (α_1, α_2) if and only if $\alpha_1 = \ell(v_1)$, $\alpha_2 = \ell(v_2)$, $v_1 \prec v_2$ in $\mathcal{CG}_D$, and there is no node v_3 with $v_1 \prec v_3 \prec v_2$ and $\ell(v_3) \in V$.*

Example 31 (continued). Consider again the match π of q in the instance I' (Figure 4, left) defined by: $\{x_1 \to a_1, x_2 \to a_2, x_3 \to t_0, x_4 \to t_1\}$. A backbone of π is shown on the right of Figure 4: $h(a_1, a_2)$ and $m(a_2)$ belong to the backbone because they each are the image of a standard atom of q; $s(a_1, t_0)$ and $r(z, t_1)$ are selected because they contain the nulls $\pi(x_3) = t_0$ and $\pi(x_4) = t_1$ (note that x_3 and x_4 only occur in path atoms of q); $b(a_1, a_2, z_0)$ is selected because it is the deepest common ancestor of $s(a_1, t_0)$ and $r(z_0, t_1)$. Then, to define the associated proof scheme induced by π, we will enrich that backbone by all the possible transitions (which is not shown on the figure).

Definition 34 (Proof scheme induced by a match). *Let π be a match of q in some I_n obtained by an $\mathcal{R}$-derivation from I. A proof scheme $\mathbb{P}_\pi$ induced by π is $(\mathcal{F} = (V, E), \mathcal{T})$, where:*

- *$\mathcal{F}$ is a backbone of π;*
- *$\mathcal{T}$ is the set of all pairs $(t, (\alpha_1, \alpha_2))$ with $(\alpha_1, \alpha_2) \in V \times V$ and $t \in \mathcal{T}_{q,I_n}(\alpha_1, \alpha_2)$, that are legal in a proof scheme, i.e., $\alpha_1 = \alpha_2$, $(\alpha_1, \alpha_2) \in E$, $(\alpha_2, \alpha_1) \in E$, or both α_1 and α_2 are roots.*

Example 31 (continued). For instance, let us assume an automaton with a single state s_r to encode r^*. The transition $((1, s_r, 2, s_r), (r(z_0, t_1), r(z_0, t_1)))$ is in $\mathbb{P}_\pi$ because *(i)* $r(z_0, t_1) = r(z_0, t_1)$ and *(ii)* the atom $r(z_0, t_1)$ belongs to chase$(I, \mathcal{R})$. It also holds that $((2, s_r, 3, s_r), (s(a_1, t_0), b(a_1, a_2, z_0))) \in \mathcal{T}$ is in $\mathbb{P}_\pi$, because *(i)* $s(a_1, t_0)$ is a child of $b(a_1, a_2, z_0)$ and *(ii)* the atom $r(t_0, z_0)$ belongs to chase$(I, \mathcal{R})$.

The transition $((1, s_r, 2, s_r), (r(t_0, z_0), r(z_0, t_1)))$ does not belong to $\mathbb{P}_\pi$, because $r(t_0, z_0)$ is neither equal to nor a parent or child of $r(z_0, t_1)$, but it belongs to the saturation of $\mathbb{P}_\pi$ because of the above two transitions. Let us note that the proof scheme induced by a match of q has a size polynomial in q. We now prove that it is indeed a proof scheme and that it is valid.

Proposition 35. *Let π be a match of q in I_n obtained by an $\mathcal{R}$-derivation D from I. Any proof scheme induced by π is valid.*

Proof. Let $\mathbb{P}_\pi = (\mathcal{F}, \mathcal{T})$ be a proof scheme induced by π. We first prove that $\mathbb{P}_\pi$ is indeed a proof scheme. Numbers refer to items of Definition 24. (1): since the chase graph $\mathcal{CG}_D$ is a directed forest, $\mathcal{F}$ is a directed forest; (2): V is by definition a set of atoms; (3): let α_1 and α_2 be two atoms of $\mathcal{F}$ containing $\bot$. By the connectivity property of $\mathcal{CG}_D$, α_1 and α_2 are labels of connected nodes in $\mathcal{CG}_D$. Let ν_1 and ν_2 be these nodes and let $\nu_3 = \text{glb}(\nu_1, \nu_2)$: by construction, $\ell(\nu_3)$ belongs to $\mathcal{F}$ and is an ancestor of ν_1 and ν_2 in $\mathcal{F}$. Hence, α_1 and α_2 are connected in $\mathcal{F}$. (4) and (5): by construction, all the transitions in the induced proof scheme are of the correct syntactic shape.

We now show that the identity is a witnessing embedding of $\mathbb{P}_\pi$ in $\mathcal{CG}_D$, proving its validity. Numbers refer to items in Definition 27. (1) By construction of V, all the root atoms are labels of roots of $\mathcal{CG}_D$, hence they belong to I. (2) By construction, $V \subseteq I_n$, and ψ is the identity, so $\psi(\alpha)$ belongs to I_n for all $\alpha \in V$. (3) By construction, for each edge (α_1, α_2) in E, $\ell^{-1}(\alpha_1) \prec \ell^{-1}(\alpha_2)$ in $\mathcal{CG}_D$, hence there is a derivation as required. (4) By construction, each $(t, (\alpha_1, \alpha_2)) \in \mathcal{T}$ belongs to $\mathcal{T}_{q,I_n}(\alpha_1, \alpha_2)$, hence to $\mathcal{T}_{q,I_n}(\psi(\alpha_1), \psi(\alpha_2))$ because ψ is the identity. We conclude that $\mathbb{P}$ is a valid proof scheme w.r.t $(I, \mathcal{R}, q)$. □

We now show that there is indeed a match of q in the (valid) proof scheme induced by a match of q in the chase.

Proposition 36. *Let $\mathbb{P}_\pi$ be a proof scheme induced by a match π of q in I_n, where I_n is the result of some derivation D from I. There exists a match of q in $\mathbb{P}_\pi$.*

Proof. We show that π is a match of q in $\mathbb{P}_\pi = (\mathcal{F} = (V, E), \mathcal{T})$.

- By definition of $\mathbb{P}_\pi$, if $\alpha \in q$ is a standard atom, then $\pi(\alpha) \in V$.
- Let $\alpha = \Lambda(x, y)$ be a path atom. As π is a match of q, there is a path in I_n from $\pi(x)$ to $\pi(y)$, whose label is recognized by the automaton associated with Λ. By definition of V (Definition 33), there exist α_x and α_y in V containing π_x and π_y. Let ν_x and ν_y s.t. $\ell(\nu_x) = \alpha_x$ and $\ell(\nu_y) = \alpha_y$. We show by induction that whenever there is a path from $\alpha_x[i]$ to $\alpha_y[j]$ in I_n that moves the automaton from s_x to s_y, then $((i_x, s_x, i_y, s_y), (\alpha_x, \alpha_y))$ is in the saturation of $\mathbb{P}$. We study the possible cases regarding the relationship between α_x and α_y in $\mathbb{P}_\pi$:
  - if $\alpha_x = \alpha_y$, or α_x is the parent of α_y (or vice-versa): the property holds by definition of the proof scheme induced by a match (Definition 34);
  - if α_x is a strict ancestor of α_y, but not a parent: we show the result by induction on the length of the path from α_x to α_y. The base case has been treated in the previous item. Let α be the child of α_x that is an ancestor of α_y. Let ν be the node of $\mathcal{CG}_D$ s.t. $\ell(\nu) = \alpha$. By the path property of $\mathcal{CG}_D$, any path that goes from x to y must go through some $z \in \text{terms}(\alpha)$ occurring at some position i_z. Let $p = p_1.p_2$, such that p_1 goes from x to z and p_2 goes from z to y. Let s_z be the state of $\mathbb{A}$ after reading p_1 starting from s_x. By induction assumption, $((i_x, s_x, i_z, s_z), (\alpha_x, \alpha))$ and $((i_z, s_z, i_y, s_y), (\alpha, \alpha_y))$ belong to the saturation of $\mathbb{P}_\pi$. Hence, $((i_x, s_x, i_y, s_y), (\alpha_x, \alpha_y))$ is in the saturation of $\mathbb{P}$;

– α_x and α_y are not in an ancestor relationship but are in the same tree. By the path property of $\mathcal{CG}_D$, any path from $\pi(x)$ to $\pi(y)$ must go through a term z of $\mathrm{glb}(v_x, v_y)$, whose label we denote by α. Let $p = p_1.p_2$ such that p_1 is a path from x to z and p_2 is a path from z to y. Let s_z be the state in which $\mathbb{A}$ is after reading p_1 starting from s_x. Assuming $z = \alpha[i_z]$, by induction assumption, $((i_x, s_x, i_z, s_z), (\alpha_x, \alpha))$ and $((i_z, s_z, i_y, s_y), (\alpha, \alpha_y))$ belong to the saturation of $\mathbb{P}_\pi$. Hence, $((i_x, s_x, i_y, s_y), (\alpha_x, \alpha_y))$ is in the saturation of $\mathbb{P}_\pi$;
– α_x and α_y are not in the same tree: let α_{R_x} (resp. α_{R_y}) be the root of the tree containing α_x (resp. α_y). By the connectivity property of $\mathcal{CG}_D$, any path from x to y must go through $\alpha_{R_x}[i_{t_x}]$ for some i_{t_x} (resp. $\alpha_{R_y}[i_{t_y}]$ for some i_{t_y}). By the preceding cases, it holds that $((i_x, s_x, i_{t_x}, s_{t_x}), (\alpha_x, \alpha_{R_x}))$ and $((i_{t_y}, s_{t_y}, i_y, s_y), (\alpha_{R_y}, \alpha_y))$ belong to the saturation of $\mathbb{P}_\pi$. By definition of $\mathbb{P}_\pi$, because α_{R_x} and α_{R_y} are roots of $\mathbb{P}_\pi$, it holds that $((i_{t_x}, s_{t_x}, i_{t_y}, s_{t_y}), (\alpha_{R_x}, \alpha_{R_y}))$ belongs to $\mathbb{P}_\pi$. Hence, by definition of the saturation of $\mathbb{P}_\pi$, $((i_x, s_x, i_y, s_y), (\alpha_x, \alpha_y))$ also belongs to the saturation of $\mathbb{P}_\pi$, which concludes the proof. □

5.3 Complexity Analysis

By Proposition 29, we know that q is entailed by I and $\mathcal{R}$ if and only if there exists a valid proof scheme $\mathbb{P}$ of polynomial size in q such that there is a match of q in $\mathbb{P}$. We now investigate how to check the validity of a proof scheme. To do so, we will consider a variant of the previous chase graph, in which we keep track of all possible ways of producing atoms. We call it the chase graph associated with $(I, \mathcal{R})$, because its set of atoms is homomorphically equivalent to $\mathrm{chase}(I, \mathcal{R})$.

Definition 37 (Chase graph associated with $(I, \mathcal{R})$). *Let I be an instance and $\mathcal{R}$ be a set of linear rules. The* chase graph *associated with $(I, \mathcal{R})$, denoted by $\mathcal{CG}(I, \mathcal{R})$, is the node-labeled directed graph defined as follows (we use ℓ for the labeling function):*

- *The restriction of ℓ to the root nodes of $\mathcal{CG}(I, \mathcal{R})$ defines a bijection between the root nodes and I.*
- *For every node v, the children of v are in bijection with the finite set of triggers (ρ_i, π_i) on $\ell(v)$, and the labels of the children are precisely the atoms α_i resulting from the application of these triggers to $\ell(v)$ (using globally unique fresh nulls).*
- *Only the nodes and edges required by the preceding two items are present in $\mathcal{CG}(I, \mathcal{R})$.*

Importantly, $\mathcal{CG}(I, \mathcal{R})$ still fulfils the connectivity property (for the same reasons as the chase graph associated with derivations). The following lemma shows how transitions between an atom in the chase graph and an ancestor atom can be computed using the transitions relating its parent atom to the same ancestor, together with the transitions that can be obtained by considering the atom in isolation.

Lemma 38. *Let v_1, v_2 and v_3 be nodes in $\mathcal{CG}(I, \mathcal{R})$ such that v_1 is a (strict) ancestor of v_2, and v_3 is a successor of v_2. Let $\alpha_i = \ell(v_i)$ for $1 \leq i \leq 3$. Then:*

(1) $(i_1, s_1, i_3, s_3) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_1, \alpha_3)$ *iff there exist* i_2, s_2, i_2' *such that* $(i_1, s_1, i_2, s_2) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_1, \alpha_2)$, $(i_2', s_2, i_3, s_3) \in \mathcal{T}_{q,\mathrm{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$ *and* $\alpha_2[i_2] = \alpha_3[i_2']$;

(2) $(i_1, s_1, i_3, s_3) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_3, \alpha_1)$ *iff there exist* i_2, s_2, i_2' *such that* $(i_1, s_1, i_2', s_2) \in \mathcal{T}_{q,\mathrm{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$, $(i_2, s_2, i_3, s_3) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_2, \alpha_1)$ *and* $\alpha_2[i_2] = \alpha_3[i_2']$;

(3) $(i_1, s_1, i_4, s_4) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_3, \alpha_3)$ *iff there exist* $i_2, s_2, i_2', i_3, s_3, i_3'$ *such that* $(i_1, s_1, i_2', s_2) \in \mathcal{T}_{q,\mathrm{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$, $(i_2, s_2, i_3, s_3) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_2, \alpha_2)$, $(i_3', s_3, i_4, s_4) \in \mathcal{T}_{q,\mathrm{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$ *and* $\alpha_3[i_2'] = \alpha_2[i_2]$ *and* $\alpha_3[i_3'] = \alpha_2[i_3]$.

Proof. The 'if' direction of the equivalences are trivial, so we concentrate on proving the 'only if' directions.

For the first statement, suppose that $(i_1, s_1, i_3, s_3) \in \mathcal{T}_{q,\mathrm{chase}(I,\mathcal{R})}(\alpha_1, \alpha_3)$. It follows that there is a path p in $\mathrm{chase}(I, \mathcal{R})$ that begins in $t_1 = \alpha_1[i_1]$, ends in $t_3 = \alpha_3[i_3]$, and is such that $\lambda(p)$ takes an automaton $\mathbb{A}$ in q from

state s_1 to state s_3. Let p_2 be the longest suffix of p that starts in $t_2 \in \mathsf{terms}(\alpha_3)$, and whose label labels a path from t_2 to t_3 in $\mathsf{chase}(\{\alpha_3\}, \mathcal{R})$. Note that such a path p_2 is guaranteed to exist, as we can always choose the empty path from t_3 to t_3. We let p_1 be such that $p = p_1.p_2$, and let s_2 be a state such $\lambda(p_1)$ takes $\mathbb{A}$ from state s_1 to s_2 and $\lambda(p_2)$ takes $\mathbb{A}$ from s_2 to s_3. We consider two cases:

- Case 1: $p = p_2$. Then $t_2 = t_1$, and t_1 must belong to α_3. By the connectivity property of $\mathcal{CG}(I, \mathcal{R})$, t_1 must also belong to α_2. There are thus indices i_2 and i_2' such that $\alpha_1[i_1] = \alpha_2[i_2] = \alpha_3[i_2']$. We thus have $(i_1, s_1, i_2, s_1) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\alpha_1, \alpha_2)$ and $(i_2', s_1, i_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$.
- Case 2: $p \neq p_2$. Let t_2' be the (unique) term that precedes t_2 in p (hence, in p_1), and let p_2' be the length-one path from t_2' to t_2 that immediately precedes p_2 in p. Clearly, $t_2 \in \mathsf{terms}(\alpha_3)$, and we claim that $t_2 \in \mathsf{terms}(\alpha_2)$. To see why, let q be the final predicate in p_1, which means the atom $q(t_2', t_2)$ occurs in $\mathsf{chase}(I, \mathcal{R})$. The connectivity property of $\mathcal{CG}(I, \mathcal{R})$ implies that t_2 must occur in all atoms that label a path between any node labeled by $q(t_2', t_2)$ and v_3. As p_2 is the longest suffix of p that is a path in $\mathsf{chase}(\{\alpha_3\}, \mathcal{R})$, we know that $q(t_2', t_2)$ does not appear in a descendant of v_3 in $\mathcal{CG}(I, \mathcal{R})$. It follows that such a path necessarily goes through v_2, hence t_2 occurs in α_2.

The second statement can be proven analogously. For the third statement, suppose that $(i_1, s_1, i_4, s_4) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\alpha_3, \alpha_3)$. Then there exists a path p in $\mathsf{chase}(I, \mathcal{R})$ that begins in $t_3 = \alpha_3[i_1]$ and ends in $t_3' = \alpha_3[i_4]$. If p is a path in $\mathsf{chase}(\{\alpha_3\}, \mathcal{R})$, then the statement trivially holds. Otherwise, let p_1 (resp. p_3) be the longest prefix (resp. longest suffix) of p ending (resp. starting) in a term t_2 (resp. t_2') of α_3 whose label labels a path from t_1 to t_2 (resp. from t_2' to t_3) in $\mathsf{chase}(\{\alpha_3\}, \mathcal{R})$. Note that since p is not a path in $\mathsf{chase}(\{\alpha_3\}, \mathcal{R})$, there exists a non-empty subpath p_2 such that $p = p_1 p_2 p_3$. Arguing as above, we can show that t_2 and t_2' must also belong to $\mathsf{terms}(\alpha_2)$. Indeed, p_2 starts with an atom that contains t_2 but does not appear in $\mathsf{chase}(\{\alpha_3\}, \mathcal{R})$, hence by the connectivity property, t_2 must appear in α_2, and similarly for the last atom of p_2 and term t_2'. We can thus find indices i_2, i_2', i_3, i_3' such that $t_2 = v_2[i_2] = \alpha_3[i_2']$ and $t_2' = v_2[i_3] = v_3[i_3']$. Putting everything together, we obtain the following: $(i_1, s_1, i_2', s_2) \in \mathcal{T}_{q,\mathsf{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$, $(i_2, s_2, i_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\alpha_2, \alpha_2)$, and $(i_3', s_3, i_4, s_4) \in \mathcal{T}_{q,\mathsf{chase}(\{\alpha_3\},\mathcal{R})}(\alpha_3, \alpha_3)$. □

We now exploit this lemma and existing complexity results on RPQ answering to design a procedure for checking validity of a proof scheme, with the following complexity:

Proposition 39. *Checking the validity of a proof scheme can be done in single exponential time, in polynomial space if predicate arity is bounded, and in NL if q and $\mathcal{R}$ are fixed.*

Proof sketch. Let $\mathbb{P} = (\mathcal{F} = (V, E), \mathcal{T})$ be a proof scheme whose validity w.r.t. $(I, \mathcal{R}, q)$ we wish to test. To establish the complexity bounds, we will show that a proof scheme is valid just in the case that some execution of the following non-deterministic procedure returns 'valid'. For the bound on the number of iterations of the inner while loop, we will use $\textsc{Max} = P * (|A| + |\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)|)^A * 2^{3*N^2*A^2}$, where P and A are respectively the number of predicates and maximum predicate arity for the ruleset $\mathcal{R}$, and N is the total number of states across all of the automata in q. (We will explain later the origin of this bound.)

Step 1: Check that every root atom in V belongs to I, and return 'not valid' if not. Next consider each pair (α, α') of (not necessarily distinct) root atoms in turn. For each possible transition $\tau = (i_1, s_1, i_2, s_2)$ w.r.t. q and (α, α'), construct the corresponding RPQ $q_\tau = \mathcal{L}_{\mathbb{A}}(s_1, s_2)(\alpha[i_1], \alpha'[i_2])$ (with $\mathbb{A}$ the automaton containing states s_1, s_2) and use the RPQ Entailment oracle to decide whether q_τ is entailed from $(I, \mathcal{R})$. Store in $\textsc{Trans}(\alpha, \alpha')$ the transitions τ for which q_τ is entailed. If there is some $(\tau, (\alpha, \alpha')) \in \mathcal{T}$ such that $\tau \notin \textsc{Trans}(\alpha, \alpha')$, return 'not valid', else continue to Step 2.

Step 2: Let Unexamined be the set of non-root nodes in V.

While Unexamined $\neq \emptyset$, choose some $\alpha_c \in$ Unexamined whose parent α_p is such that $\alpha_p \notin$ Unexamined, and proceed as follows:

- Remove α_c from Unexamined.
- Set Count = 0 and $\alpha_i = \alpha_p$.
- While Count $\leq$ Max and $\alpha_i \neq \alpha_c$
  (a) Guess a rule $\rho \in \mathcal{R}$ that is applicable to α_i using substitution σ (return 'not valid' if no such rule).
  (b) Let α_i' be the result of applying ρ to α_i under σ. We assume that existential variables in the head of ρ are replaced with either nulls from $\mathsf{terms}(\alpha_c) \setminus (\mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_i))$ or fresh nulls (not in $\mathsf{terms}(\mathcal{F})$) using the next available ids.
  (c) For each possible transition $\tau = (k_1, s_1, k_2, s_2)$ w.r.t. q and (α_i', α_i')
    – construct the RPQ $q_\tau = \mathcal{L}_{\mathbb{A}}(s_1, s_2)(\alpha_i'[k_1], \alpha_i'[k_2])$ (with $\mathbb{A}$ the automaton containing states s_1, s_2)
    – use the RPQ Entailment oracle to decide if q_τ is entailed from $(\{\alpha_i'\}, \mathcal{R})$
    – if q_τ is entailed, then store τ in $\textsc{Trans}^{\downarrow}(\alpha_i', \alpha_i')$
  (d) For every possible transition $\tau = (k_1, s_1, k_3, s_3)$ w.r.t. q and (α_p, α_i'), add τ to $\textsc{Trans}(\alpha_p, \alpha_i')$ if there exist transitions $(k_1, s_1, k_2, s_2) \in \textsc{Trans}(\alpha_p, \alpha_i)$ and $(k_2', s_2, k_3, s_3) \in \textsc{Trans}^{\downarrow}(\alpha_i', \alpha_i')$ such that $\alpha_i[k_2] = \alpha_i'[k_2']$
  (e) For every possible transition $\tau = (k_1, s_1, k_3, s_3)$ w.r.t. q and (α_i', α_p), add τ to $\textsc{Trans}(\alpha_i', \alpha_p)$ if there exist transitions $(k_1, s_1, k_2', s_2) \in \textsc{Trans}^{\downarrow}(\alpha_i', \alpha_i')$ and $(k_2, s_2, k_3, s_3) \in \textsc{Trans}(\alpha_i, \alpha_p)$ such that $\alpha_i'[k_2'] = \alpha_i[k_2]$
  (f) For every possible transition $\tau = (k_1, s_1, k_4, s_4)$ w.r.t. q and (α_i', α_i'), add τ to $\textsc{Trans}(\alpha_i', \alpha_i')$ whenever there exist transitions $(k_1, s_1, k_2', s_2) \in \textsc{Trans}^{\downarrow}(\alpha_i', \alpha_i')$, $(k_2, s_2, k_3, s_3) \in \textsc{Trans}(\alpha_i, \alpha_i)$, and $(k_3', s_3, k_4, s_4) \in \textsc{Trans}^{\downarrow}(\alpha_i', \alpha_i')$ such that $\alpha_i'[k_2'] = \alpha_i[k_2]$ and $\alpha_i'[k_3'] = \alpha_i[k_3]$
  (g) Set $\alpha_i = \alpha_i'$ and increment Count.
- Return 'not valid' if one of the following conditions does not hold:
  – $\alpha_i = \alpha_c$
  – $\{\tau \mid (\tau, (\alpha_c, \alpha_c)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_c, \alpha_c) = \textsc{Trans}(\alpha_i, \alpha_i)$
  – $\{\tau \mid (\tau, (\alpha_p, \alpha_c)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_p, \alpha_c) = \textsc{Trans}(\alpha_p, \alpha_i)$
  – $\{\tau \mid (\tau, (\alpha_c, \alpha_p)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_c, \alpha_p) = \textsc{Trans}(\alpha_i, \alpha_p)$

Return 'valid'.

We remark that the preceding procedure runs in polynomial space, except that it makes some calls to an oracle for RPQ Entailment. As RPQ entailment is in ExpTime and PSpace = NPSpace $\subseteq$ Exptime, the procedure can be made to run in Exptime. If we consider rules of bounded arity, then RPQ entailment is in PSpace, so the procedure runs in polynomial space. Finally, let us consider the case where q and $\mathcal{R}$ are fixed and show that the procedure can be made to run in non-deterministic logarithmic space. For Step 1, it suffices to iterate over the polynomially many combinations of possible transitions and pairs of root atoms, and for every such combination, test whether the corresponding RPQ is entailed. As RPQ Entailment is in NLogSpace in data complexity, and it is known that $\text{NLogSpace}^{\text{NLogSpace}} \subseteq$ NLogSpace, Step 1 can be implemented in NLogSpace. For Step 2, by proceeding in a depth-first manner, we can examine each of the nodes in the proof scheme using only logarithmic space to keep track of which nodes remain to be visited. For each node α_c with parent α_p, the maximal number of iterations of the inner while loop is bounded by a constant (as it depends only on q and $\mathcal{R}$). Moreover, it is easily verified that all operations in the inner while loop can be performed using constant space, except for Step 2(c), which involves a constant number of RPQ Entailment checks, each of which can be handled by an NLogSpace oracle. Thus, the entire procedure can be implemented in non-deterministic logspace, yielding the desired NLogSpace upper bound. We have thus shown that the procedure gives the required complexity bounds, and in the appendix, we establish its correctness. □

Note that the proposition can be extended to arbitrary linear rules with multiple head atoms (translated into atomic-head linear rules). Indeed, for combined complexity in the bounded predicate arity case, we know that RPQ entailment remains in PTime, see Proposition 17 and Theorem 20; hence, checking the validity of a proof scheme remains in PSpace.

It follows from Proposition 39 that enumerating polysize proof schemes and checking for their validity and for the existence of a match of q in them yields a sound and complete algorithm for CRPQ answering.

Theorem 40. *CRPQ answering under linear rules is ExpTime-complete in combined complexity, PSpace-complete in combined complexity with bounded-predicate arity, and NL-complete in data complexity.*

Proof. By Proposition 29, CRPQ entailment comes down to deciding whether there exists a polynomial (in the query) proof scheme that is valid and contains a match for the query.

Though polysize proof schemes are defined on an infinite number of atoms, and are thus infinite in number, we can enumerate in exponential time those that are different up to a renaming of nulls. Thus, for the general case, we can enumerate all polysize proof schemes that are distinct up to a renaming of nulls in polynomial space (and exponential time) and check whether there exists a valid proof scheme that contains a match for the query. As already noted, deciding existence of a match in a proof scheme is in NP, and by Proposition 39, validity checking is in ExpTime. Thus, the described procedure runs in single-exponential time. When the arity is bounded, validity checking is in PSpace (by Proposition 39), and so the overall procedure runs in polynomial space. Finally, if both q and $\mathcal{R}$ are fixed, the number of relevant proof schemes and their size are constants. Thus, we can enumerate all proof schemes in constant time, and checking for a match is also in constant time. By Proposition 39, the validity checking is in NL w.r.t. the size of the data, so the overall procedure runs in non-deterministic logspace for data complexity.

The ExpTime lower bound in combined complexity and the NL lower bound in data complexity are immediate consequence of the ExpTime-hardness (resp. NL-hardness) of RPQ answering for linear rules w.r.t. combined (resp. data) complexity. The PSpace lower bound in the bounded arity case is inherited from the PSpace-hardness of CRPQ answering over DL-Lite$_\mathcal{R}$ knowledge bases (Bienvenu et al. 2015). □

6 CRPQ Answering under Guarded Rules: Upper Bound

To upper bound the complexity of CRPQ answering under guarded rules, we reduce this problem to CRPQ answering under linear rules. For simplicity, we assume that all the predicates in I also occur in $\mathcal{R}$ (however, the reduction is easily extended to drop this restriction).

Two main ideas underly the reduction. First, we define an alternative notion of derivation, called 'locally complete' derivation, which ensures the following property: at each step k of the derivation, the considered extended instance $\hat{I}_k$ contains exactly the subset of the chase restricted to its terms, *i.e.*, $\text{chase}(I,\mathcal{R})|_{\text{terms}(\hat{I}_k)}$. Of course such a derivation is not computable for an arbitrary set of rules, because it requires the decidability of (atom) entailment, but it is for guarded rules. Second, the application of guarded rules (in a locally complete derivation) is simulated through linear rules (in a classical derivation) expressed on an extended vocabulary, in which each predicate represents a guarded set of atoms. We use a finite set of canonical variables to rename terms occurring in the chase atoms. This ensures there are finitely many guarded sets to be considered.

Let us first formally introduce the notion of locally complete derivation. A locally complete derivation is a sequence of extended instances $\hat{I}_i$ that have the property of being closed with respect to the atoms on $\text{terms}(\hat{I}_i)$ that can be derived from $\hat{I}_i$. Starting from an instance I, $\hat{I}_0$ is obtained from I by closing it under atomic entailment. At each step i, a rule is applied on $\hat{I}_i$ and the resulting set is again closed to yield $\hat{I}_{i+1}$.

Definition 41 (Locally complete derivation). *Let I be an instance and $\mathcal{R}$ be a set of existential rules. A* locally complete $\mathcal{R}$-derivation *from I resulting in $\hat{I}_n$ is a sequence $\hat{I}_0, \hat{I}_1, \ldots, \hat{I}_n$ with $\hat{I}_0 = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}$ and such that for all $i \geq 0$, there are $\rho_i \in \mathcal{R}$ and a homomorphism π_i from $\mathsf{body}(\rho_i)$ to $\hat{I}_i$ such that $\hat{I}_{i+1} = \mathsf{chase}(\hat{I}'_i, \mathcal{R})_{|\mathsf{terms}(\hat{I}'_i)}$ where $\hat{I}'_i = \hat{I}_i \cup \pi_i^{\mathrm{safe}}(\mathsf{head}(\rho_i))$.*

Note that a locally complete derivation could have been defined as a possibly infinite sequence, but we will only need finite sequences to state our results. Next, we show that (finite) classical derivations and locally complete derivations are equivalent for our purposes, meaning that they entail the same Boolean CRPQs. This result holds more generally for homomorphism-closed queries, as stated in Proposition 45.

Let us first illustrate the notion of locally complete derivation on the running example.

Example 42 (Running Example—continued). *Let $\mathcal{R} = \{\rho_1, \ldots, \rho_6\}$ be the considered set of rules, which we recall below for reading convenience.*

(ρ_1) $\mathsf{isFriendOf}(x, y) \rightarrow \mathsf{isFriendOf}(y, x)$
(ρ_2) $\mathsf{isFriendOf}(x, y) \rightarrow \mathsf{follows}(x, y)$
(ρ_3) $\mathsf{follows}(x, y) \rightarrow \exists m\, \mathsf{message}(m, x, y)$
(ρ_4) $\mathsf{follows}(x, y) \wedge \mathsf{follows}(y, x) \rightarrow \mathsf{isPaired}(x, y)$
(ρ_5) $\mathsf{message}(m, x, y) \rightarrow \mathsf{sends}(x, m)$
(ρ_6) $\mathsf{message}(m, x, y) \rightarrow \mathsf{receives}(y, m)$

Consider again the instance $I = \{\mathsf{follows}(\mathsf{B}, \mathsf{A}), \mathsf{isFriendOf}(\mathsf{C}, \mathsf{A}), \mathsf{isFriendOf}(\mathsf{C}, \mathsf{B})\}$. Then, $\hat{I}_0$ is obtained from I by adding all the entailed facts on terms in $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$:

$\mathsf{isFriendOf}(\mathsf{A}, \mathsf{C}), \mathsf{isFriendOf}(\mathsf{B}, \mathsf{C}),$
$\mathsf{follows}(\mathsf{C}, \mathsf{A}), \mathsf{follows}(\mathsf{C}, \mathsf{B}), \mathsf{follows}(\mathsf{A}, \mathsf{C}), \mathsf{follows}(\mathsf{B}, \mathsf{C}),$
$\mathsf{isPaired}(\mathsf{C}, \mathsf{A}), \mathsf{isPaired}(\mathsf{A}, \mathsf{C}), \mathsf{isPaired}(\mathsf{C}, \mathsf{B}), \mathsf{isPaired}(\mathsf{B}, \mathsf{C}).$

In a classical derivation, $\hat{I}_0$ would be obtained from I by triggering the rules ρ_1, ρ_2 and ρ_4. Now, assume Rule ρ_3 is applied on $\hat{I}_0$ by homomorphism $\pi = \{x \mapsto \mathsf{B}, y \mapsto \mathsf{A}\}$. Then, $I'_1 = \hat{I}_0 \cup \{\mathsf{message}(\mathsf{m}_0, \mathsf{B}, \mathsf{A})\}$, and $\hat{I}_1 = I'_1 \cup \{\mathsf{sends}(\mathsf{B}, m_0), \mathsf{receives}(\mathsf{A}, m_0)\}$. In a classical derivation, $\hat{I}_1$ would be obtained from I'_1 by triggering the rules ρ_5 and ρ_6. The locally complete derivation can be continued by applying ρ_3 on the other facts with predicate follows.

Let us outline the main ideas of the reduction presented next. We define a finite set of complex predicates*, which are canonical encodings of all the guarded sets on a vocabulary. Then, each atom α in I is replaced by a complex atom (i.e., an atom with a complex predicate) encoding α and its guarded set in $\hat{I}_0$. For example, $\alpha = \mathsf{isFriendOf}(\mathsf{C}, \mathsf{A})$ is replaced by a complex atom encoding α and the following set:*

$$\{\mathsf{isFriendOf}(\mathsf{C}, \mathsf{A}), \mathsf{isFriendOf}(\mathsf{A}, \mathsf{C}), \mathsf{follows}(\mathsf{C}, \mathsf{A}), \mathsf{follows}(\mathsf{A}, \mathsf{C}), \mathsf{isPaired}(\mathsf{C}, \mathsf{A}), \mathsf{isPaired}(\mathsf{A}, \mathsf{C})\}$$

Furthermore, each rule from $\mathcal{R}$ is replaced by a set of complex rules*, which are linear rules using complex predicates, such that each application of a complex rule simulates a step of a locally complex derivation. Finally,* reconstruction rules *translate back atoms on the complex predicates into atoms on the initial predicates. We will illustrate the multiple intricacies of this translation using specific technical examples.*

The following Lemmas 43 and 44 allow us to show that classical derivations and locally complete derivations are able to derive exactly the same conclusions.

Lemma 43. *For any classical $\mathcal{R}$-derivation from I to I_n, there exists a locally complete $\mathcal{R}$-derivation from I to $\hat{I}_m$ such that $\hat{I}_m \models I_n$.*

Proof. We prove the result by induction on the length of the classical derivation. If $n = 0$, the result holds since $I_0 \subseteq \hat{I}_0$. Otherwise, let us assume that the result holds for any classical derivation of length $n \geq 0$. Let

$I = I_0, \ldots, I_n, I_{n+1}$ be a classical derivation of length $n+1$, and ρ_n and π_n be such that I_{n+1} is obtained from I_n by applying the trigger (ρ_n, π_n). By induction assumption, there is a locally complete $\mathcal{R}$-derivation from I resulting in $\hat{I}_m$ such that $\hat{I}_m \models I_n$. Hence, there is a homomorphism h_n from I_n to $\hat{I}_m$. Thus $(\rho_n, h_n \circ \pi_n)$ is a trigger on $\hat{I}_m$, and h_n can be extended to a homomorphism from I_{n+1} to $\hat{I}'_{m+1} = \hat{I}_m \cup (h_n \circ \pi_n)^{\text{safe}}(\text{head}(\rho_n))$ by mapping the existential variables in $\pi_n^{\text{safe}}(\text{head}(\rho_n))$ to the corresponding existential variables in $(h_n \circ \pi_n)^{\text{safe}}(\text{head}(\rho_n))$. Hence, the locally complete derivation from I resulting in $\hat{I}_m$ can be extended to a locally complete derivation from I resulting in $\hat{I}_{m+1} = \text{chase}(\hat{I}'_{m+1}, \mathcal{R})_{|\text{terms}(\hat{I}'_{m+1})}$, and $\hat{I}_{m+1} \models \hat{I}'_{m+1} \models I_{n+1}$, which concludes this proof. □

Lemma 44. *For any locally complete $\mathcal{R}$-derivation from I resulting in $\hat{I}_n$, there exists a classical $\mathcal{R}$-derivation from I resulting in I' such that $I' \models \hat{I}_n$.*

Proof. We prove the result by induction on the length of a locally complete derivation $\hat{I}_0, \ldots, \hat{I}_n$. For $n = 0$, $\hat{I}_n = \text{chase}(I, \mathcal{R})_{|\text{terms}(I)}$. As $\text{chase}(I, \mathcal{R}) \models \text{chase}(I, \mathcal{R})_{|\text{terms}(I)}$, there is I_m derivable from I such that $I_m \models \hat{I}_n$, which concludes that case. For the induction step, let us assume the result for any locally complete derivation of length up to $n \geq 0$, and let $\hat{I}_{n+1}$ be obtained from a locally complete derivation of length $n+1$. We obtained $\hat{I}_{n+1}$ from $\hat{I}_n$ by a single locally complete rule application of rule ρ_n through π_n. Let $I = I_0, \ldots, I_m$ be a derivation such that $I_m \models \hat{I}_n$, and φ_n be a homomorphism from $\hat{I}_n$ to I_m. Then ρ_n is applicable to I_m using $\varphi_n \circ \pi_n$, creating I_{m+1}. Let φ_n^+ be the extension of φ_n such that each variable introduced by the application of ρ_n to $\hat{I}_n$ through π_n is mapped to the corresponding variable in I_{m+1}. We show that there exist an extension $I_0, \ldots, I_{m+1}, \ldots, I_{m'}$ of the derivation $I_0, \ldots, I_{m+1}$ and a homomorphism φ_{n+1} from $\hat{I}_{n+1}$ to $I_{m'}$.

First observe that φ_n^+ is a homomorphism from $\hat{I}'_n = \hat{I}_n \cup \pi_n^{\text{safe}}(\text{head}(\rho_n))$ to I_{m+1}. By definition, $\hat{I}_{n+1} = \text{chase}(\hat{I}'_n, \mathcal{R})_{|\text{terms}(\hat{I}'_n)}$. There is thus a sequence of rule applications $\{(\rho'_i, \pi'_i)\}_{1 \leq i \leq l}$ starting from $\hat{I}'_n$ whose result entails $\hat{I}_{n+1}$; let us denote by $\hat{I'}_n^i$ the set of atoms obtained after the application of (ρ'_i, π'_i). We build by induction on l a sequence of pairs (φ^i, I_{m+1+i}) such that φ^i is a homomorphism from $\hat{I'}_n^i$ to I_{m+1+i}:

- If the sequence of rule applications is empty, one can take φ_n^+ and I_{m+1}.
- If we have built φ^i and I_{m+1+i} fulfilling the induction condition, then ρ'_{i+1} is applicable to I_{m+1+i} by $\varphi^i \circ \pi'_{i+1}$, and let us denote by $I_{m+1+i+1}$ the result of that application. Then φ^i can be extended to a homomorphism φ^{i+1} from $\hat{I'}_n^{i+1}$ to $I_{m+1+i+1}$, by mapping the introduced fresh nulls in $\hat{I'}_n^{i+1}$ to the corresponding fresh nulls in $I_{m+1+i+1}$.

We finally define $I_{m'}$ as I_{m+l} and φ_{n+1} as φ^l, and we have that $I_{m'} \models \hat{I}_{n+1}$, as φ_{n+1} is a homomorphism from $\hat{I}_{n+1}$ to $I_{m'}$. □

Proposition 45. *For any KB $(I, \mathcal{R})$ and homomorphism-closed Boolean query q, there is a (classical) $\mathcal{R}$-derivation from I resulting in I_n with $I_n \models q$ if and only if there is a locally complete $\mathcal{R}$-derivation from I resulting in $\hat{I}_m$ with $\hat{I}_m \models q$.*

Proof. This is a direct consequence of Lemmas 43 and 44. □

The previous correspondence between classical and locally complete derivations holds for arbitrary instances I and rulesets $\mathcal{R}$. However, even $\hat{I}_0$ is not computable in the general case (whereas each $\hat{I}_i$ is computable in the case of guarded rules). Next, we want to simulate locally complete derivations under guarded rules by classical derivations under linear rules. To that end, we will finitely encode the infinite set of all guarded sets that can be defined on a KB vocabulary $\mathcal{V} = (\mathcal{P}, \mathcal{C})$. Note that a guarded set itself is necessarily finite because only a finite set of atoms can be defined on a finite set of terms (those of the guard) and a finite set of predicates (as $\mathcal{P}$ is finite).

To finitely encode all guarded sets, we first rename them in a canonical way. We consider a new set of *canonical variables* $\mathcal{X} = \{X_1, \ldots, X_{2w}\}$, where w is the maximal predicate arity in $\mathcal{V}$ (then, $2w$ is be the maximal number

of variables in a guarded rule and this will also hold for the linear rules built by the reduction). The set $\mathcal{X}$ is linearly ordered according to $1 \dots 2w$. The canonical renaming of an atom on $\mathcal{V}$ is obtained by substituting all its terms by canonical variables. We will need to consider atoms in which canonical variables from a subset $\mathcal{Y} \subseteq \mathcal{X}$ may already appear. In this case, its canonical renaming is obtained by substituting each other term by the next variable in $\mathcal{X} \setminus \mathcal{Y}$.

Definition 46 (Canonical renaming). *Let α be an atom that may contain both classical terms and canonical variables from $\mathcal{X}$. The* canonical renaming *of α w.r.t $\mathcal{Y} \subseteq \mathcal{X}$ is a substitution $\Phi_{\alpha,\mathcal{Y}}$ from $\mathsf{terms}(\alpha)$ to $\mathcal{X}$ such that $\Phi_{\alpha,\mathcal{Y}}(t_i) = t_i$ if $t_i \in \mathcal{X}$, otherwise $\Phi_{\alpha,\mathcal{Y}}(t_i) = X_j$ where j is the smallest integer such that $X_j \notin \mathcal{Y}$, X_j does not occur in α, and $X_j \neq \Phi_{\alpha,\mathcal{Y}}(t_k)$ for all $k < i$. If $\mathcal{Y} = \emptyset$, we simply write Φ_α.*

Note that, for a guarded set G with guard α (we recall that such set is denoted by the pair (G, α)), the canonical renaming of α yields a renaming of all terms in G.

Given a vocabulary $\mathcal{V} = (\mathcal{P}, C)$, we denote by $\mathcal{G}$ the (finite) set of all guarded sets of atoms with predicates in $\mathcal{P}$ and terms in the canonical set of variables $\mathcal{X}$. For any guarded set (G, α) on $\mathcal{V}$, it holds that that $(\Phi_\alpha(G), \Phi_\alpha(\alpha)) \in \mathcal{G}$. Furthermore, to each guarded set (G, α) we assign a *complex predicate*, of the form $p_{\Phi_\alpha(G)}$ with the same arity as α. As $\mathcal{G}$ is finite, so is the set of complex predicates. A *complex atom* is an atom with a complex predicate (and classical terms). The *canonical atom* associated with a guarded set (G, α) is $\mathsf{can}(G, \alpha) = p_{\Phi_\alpha(G)}(\mathsf{terms}(\alpha))$.

In this way, an instance can be encoded by the canonical atoms associated with all the guarded sets of its atoms. Similarly, a guarded rule is encoded by two canonical atoms respectively associated with its (guarded) body and head, which yields a linear rule. However, as illustrated by the next example, additional information will be needed to mimic guarded rule derivations with linear rule derivations.

Example 47. *Consider $I = \{p(a), r(a,b), r(b,c), q(c)\}$ and $\mathcal{R} = \{r(x,y) \wedge q(y) \rightarrow q(x); p(x) \wedge q(x) \rightarrow \exists y\, s(x,y); q(x) \wedge s(x,y) \rightarrow h(y)\}$. The way we intend to encode the instance I works as follows. Consider $r(a,b)$ for instance. It guards the set of atoms $\{p(a), r(a,b)\}$. As such, we would encode it by $p_{\{r(X_1,X_2),p(X_1)\}}(a,b)$. Similarly, we would also encode the set $\{r(b,c), q(c)\}$ guarded by $r(b,c)$ through the atom $p_{\{r(X_1,X_2),q(X_2)\}}(b,c)$.*

One could try to simulate the effect of $r(x,y) \wedge q(y) \rightarrow q(x)$ by linear rules of the shape $p_{\{r(X_1,X_2),q(X_2)\}}(x,y) \rightarrow p_{\{r(X_1,X_2),q(X_2),q(X_1)\}}(x,y)$. The rule $r(x,y) \wedge q(y) \rightarrow q(x)$ is applicable on I by mapping x to b and y to c producing $q(b)$; similarly, $p_{\{r(X_1,X_2),q(X_2)\}}(x,y) \rightarrow p_{\{r(X_1,X_2),q(X_2),q(X_1)\}}(x,y)$ is applicable on the encoded instance by mapping x to b and y to c, producing the atom $p_{\{r(X_1,X_2),q(X_2),q(X_1)\}}(b,c)$. However, the next application of the same rule, mapping x to a and y to b, cannot be simulated in the translation, because $q(b)$ is not encoded in the atom $p_{\{r(X_1,X_2),p(X_1)\}}(a,b)$, which prevents further rule applications.

As witnessed by Example 47, starting from the guarded sets present in I is not enough to mimic guarded rule derivations by linear rule derivations. A first idea would be to start from saturated guarded sets, as supported by the following proposition (whose proof is given in the appendix).

Proposition 48. *Let $\mathcal{R}$ be a set of guarded rules and I be an instance. For each atom $\alpha \in I$, we note $I^*_\alpha = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(\alpha)}$. Then:*

(⇒) *For any $\mathcal{R}$-derivation from I to I_n, we can define for every $\alpha \in I$ an $\mathcal{R}$-derivation from I^*_α to I'_α such that $\bigcup_{\alpha \in I} I'_\alpha \models I_n$;*

(⇐) *Let for any $\alpha \in I$ an $\mathcal{R}$-derivation from I^*_α to I'_α; there exists an $\mathcal{R}$-derivation from I to I' such that $I' \models \bigcup_{\alpha \in I} I'_\alpha$.*

However, a similar problem of incompleteness would occur if the (extended) instance resulting from a rule application was not closed as well. That is why the notion of locally complete derivation considers at each step the chase of the resulting instance restricted to the terms of this instance.

We are now ready to define the reduction itself. Given a KB $(I, \mathcal{R})$ on a vocabulary $\mathcal{V}$ (the 'original vocabulary'), where $\mathcal{R}$ is a set of guarded rules, we build $(I', \mathcal{R}')$ where $\mathcal{R}'$ is a set of linear rules, such that for any Boolean CRPQ

q on $\mathcal{V}$, it holds that $I, \mathcal{R} \models q$ if and only if $I', \mathcal{R}' \models q$. The new instance I' is the encoding of $\mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}$ using complex predicates, while $\mathcal{R}'$ contains two kinds of rules: *reconstruction rules*, which allow to generate back atoms on $\mathcal{V}$ from atoms on complex predicates, and *complex rules*, which simulate one step of a locally complete derivation. Let us start with the definition of I' and its illustration.

Definition 49 (Guarded Translation of I (I')). *The* guarded translation *of I w.r.t. $\mathcal{R}$ is $I' = \{\mathsf{can}(I^*_\alpha, \alpha) \mid \alpha \in I\}$, where (I^*_α, α) denotes the guarded set $(\mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(\alpha)}, \alpha)$.*

Example 50. *Consider I and $\mathcal{R}$ from Example 47, and let $\alpha_1 = r(b, c)$ and $\alpha_2 = r(a, b)$. In the following, we underline the guard in a guarded set. $(I^*_{\alpha_1}, \alpha_1) = \{\underline{r(b,c)}, q(b), q(c)\}$, hence $\mathsf{can}(I^*_{\alpha_1}, \alpha_1) = p_{\{\underline{r(X_1,X_2)}, q(X_1), q(X_2)\}}(b, c)$. Similarly, $(I^*_{\alpha_2}, \alpha_2) = \{\underline{r(a,b)}, p(a), q(a), q(b)\}$, hence $\mathsf{can}(I^*_{\alpha_2}, \alpha_2) = p_{\{\underline{r(X_1,X_2)}, p(X_1), q(X_1), q(X_2)\}}(a, b)$.*

We now present the set of linear rules $\mathcal{R}' = \mathcal{R}_r \cup \mathcal{R}_c$, which simulates the locally complete chase, starting with reconstruction rules $\mathcal{R}_r$.

Definition 51 (Reconstruction rules ($\mathcal{R}_r$)). *Let $(G, \alpha) \in \mathcal{G}$. The set of* reconstruction rules *associated with (G, α) contains for each $\beta \in G$ the rule of the form $\mathsf{can}(G, \alpha) \to \beta$. The set of all reconstruction rules (associated with $\mathcal{G}$) is denoted by $\mathcal{R}_r$.*

Note that reconstruction rules do not contain any existentially quantified variable.

Example 52. *Consider the guarded set $\{\underline{r(X_1, X_2)}, p(X_2)\}$. There are two associated reconstruction rules, namely $p_{\{\underline{r(X_1,X_2)}, p(X_2)\}}(X_1, X_2) \to r(X_1, X_2)$ and $p_{\{\underline{r(X_1,X_2)}, p(X_2)\}}(X_1, X_2) \to p(X_2)$.*

Their role is to reconstruct the atoms on the original vocabulary that are encoded by complex predicates.

Definition 53 (Expansion). *The* expansion *of a complex atom α, denoted by* $\mathsf{expansion}(\alpha)$*, is defined as $\mathsf{chase}(\alpha, \mathcal{R}_r) \setminus \{\alpha\}$. The expansion of a set of complex atoms is the union of the expansions of its atoms.*

The next lemmas formalize the role of reconstruction rules.

Lemma 54. *Let (G, α) be a guarded set of atoms. It holds that*

$$\mathsf{expansion}(\mathsf{can}(G, \alpha)) = G.$$

Proof. We prove both inclusions. Let us first prove that $G \subseteq \mathsf{expansion}(\mathsf{can}(G, \alpha))$. Let $\beta \in G$. It holds that $\Phi_\alpha(\beta) \in \Phi_\alpha(G)$. Moreover $\mathsf{can}(\Phi_\alpha(G), \Phi_\alpha(\alpha)) \to \Phi_\alpha(\beta)$ belongs to $\mathcal{R}_r$ as $(\Phi_\alpha(G), \Phi_\alpha(\alpha)) \in \mathcal{G}$. This rule is applicable to $\mathsf{can}(G, \alpha)$ and generates β.

We now prove that $\mathsf{expansion}(\mathsf{can}(G, \alpha)) \subseteq G$. Any reconstruction rule applicable to $\mathsf{can}(G, \alpha)$ is of the form $\mathsf{can}(\Phi_\alpha(G), \Phi_\alpha(\alpha)) \to \Phi_\alpha(\gamma)$ for some $\gamma \in G$. Applying such a rule to $\mathsf{can}(G, \alpha)$ generates β only if $\beta \in G$. □

Using Lemma 54, we show that the expansion of I' is the first instance of a locally complete $\mathcal{R}$-derivation of I.

Lemma 55. *It holds that:*

$$\mathsf{expansion}(I') = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}.$$

Proof. By definition, $I' = \cup_{\alpha \in I} \mathsf{can}(I^*_\alpha, \alpha)$. Hence, $\mathsf{expansion}(I') = \cup_{\alpha \in I} \mathsf{expansion}(\mathsf{can}(I^*_\alpha, \alpha))$. By applying Lemma 54, it holds that $\mathsf{expansion}(\mathsf{can}(I^*_\alpha, \alpha)) = I^*_\alpha$, hence $\mathsf{expansion}(I') = \cup_{\alpha \in I} I^*_\alpha$. As $\cup_{\alpha \in I} I^*_\alpha = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}$ by Proposition 48, this concludes the proof. □

We now focus on complex rules. We recall that their role is to simulate one step of a locally complete derivation.

Definition 56 (Complex rules ($\mathcal{R}_c$)). *Let $(G, \alpha) \in \mathcal{G}$ and $\rho \in \mathcal{R}$ be applicable to G by π. Let $\alpha' = \pi^{\text{safe}}(\text{head}(\rho))$ and $\Phi_{\alpha',\text{terms}(\alpha)}(\alpha')$ be the canonical renaming of α' w.r.t.* $\text{terms}(\alpha)$. *The* complex rule *associated with G, ρ and π is:*

$$\text{can}(G, \alpha) \rightarrow \text{can}(G', \Phi_{\alpha',\text{terms}(\alpha)}(\alpha'))$$

where $G' = \Phi_{\alpha',\text{terms}(\alpha)}(\text{chase}(G \cup \{\alpha'\}, \mathcal{R})_{|\text{terms}(\alpha')})$. The set of complex rules (associated with $\mathcal{G}$) is denoted by $\mathcal{R}_c$.

Example 57. *We continue Example 47. Consider the guarded set $G = \{\underline{r(X_1, X_2)}, p(X_1), q(X_1), q(X_2)\}$. The rule $p(x) \wedge q(x) \rightarrow \exists y\, s(x, y)$ is applicable to G through $\pi = \{x \mapsto X_1\}$. Hence, we build the complex rule $p_{\underline{r(X_1,X_2)},p(X_1),q(X_1),q(X_2)}(X_1, X_2) \rightarrow p_{G'}(X_1, X_3)$, where $G' = \{\underline{s(X_1, X_3)}, p(X_1), q(X_1), h(X_3)\}$. Note that G' contains $\underline{h(X_3)}$, which is not justified by the application of $p(x) \wedge q(x) \rightarrow \exists y\, s(x, y)$ itself, but by the future application of $q(x) \wedge s(x, y) \rightarrow h(y)$.*

We now focus on showing the correspondence between a (classical) $\mathcal{R}_c$-derivation and a locally complete $\mathcal{R}$-derivation. Lemma 58 specifies the correspondence at the level of a single rule application.

Lemma 58. *Let $(G, \alpha = r(\mathbf{t}))$ be a guarded set such that $\text{chase}(G, \mathcal{R})_{|\text{terms}(G)} = G$. The following statements hold:*

- *Let $\rho \in \mathcal{R}$ be applicable to G by π, and let $\hat{G}$ be obtained by the corresponding locally complete derivation step. There exists a complex rule ρ' applicable to $p_{\Phi_\alpha(G)}(\mathbf{t})$ by π' such that there exists a homomorphism φ from $\hat{G}$ to $\text{expansion}(p_{\Phi_\alpha(G)}(\mathbf{t}) \cup \pi'^{\text{safe}}(\text{head}(\rho')))$ with φ being the identity on $\text{terms}(G)$.*
- *Let $\rho' \in \mathcal{R}_c$ be applicable to $G' = p_{\Phi_\alpha(G)}(\mathbf{t})$ by π', and let $F' = G' \cup \pi'^{\text{safe}}(\text{head}(\rho'))$. There exists $\rho \in \mathcal{R}$ applicable to G, generating $\hat{G}$ by one step of locally complete derivation, such that there exists a homomorphism φ from $\text{expansion}(F')$ to $\hat{G}$ with φ being the identity on $\text{terms}(G)$.*

Proof. (1) Let ρ be applicable to G by π, generating $\hat{\alpha}$. Then ρ is applicable to $\Phi_\alpha(G)$ by $\pi^* = \Phi_\alpha \circ \pi$. By the definition of complex rules, there is a rule $\rho' = \text{can}(\Phi_\alpha(G), \Phi_\alpha(\alpha)) \rightarrow \text{can}(G^*, \Phi_{\alpha^*,\text{terms}(\Phi_\alpha(\alpha))}(\alpha^*))$, where we define $\alpha^* = \pi^{*\text{safe}}(\text{head}(\rho))$ and $G^* = \Phi_{\alpha^*,\text{terms}(\Phi_\alpha(\alpha))}(\text{chase}(\Phi_\alpha(G) \cup \{\alpha^*\}, \mathcal{R})_{|\text{terms}(\alpha^*)})$. The rule ρ' is applicable to $\text{can}(G, \alpha) = p_{\Phi_\alpha(G)}(\mathbf{t})$ by $\pi' = \Phi_\alpha^{-1}$ and yields the atom $\alpha' = \pi'^{\text{safe}}(\text{head}(\rho'))$. Let $\hat{G}$ be the result of performing a single locally complete derivation step to G using the rule ρ and π, i.e., $\hat{G} = \text{chase}(G \cup \hat{\alpha}, \mathcal{R})_{|\text{terms}(G \cup \{\hat{\alpha}\})}$. Define φ as the injective mapping that is the identity on the terms of G and that associates with each fresh null in $\hat{\alpha}$ the corresponding fresh null from α' that was introduced by applying ρ' to $p_{\Phi_\alpha(G)}(\mathbf{t})$.

We claim that φ is a homomorphism from $\hat{G}$ to $\text{expansion}(\text{can}(G, \alpha) \cup \alpha')$. Indeed, if $\beta \in \hat{G}$ and $\text{terms}(\beta) \subseteq \text{terms}(G)$, then by assumption, $\beta \in G$, and thus belongs to $\text{expansion}(\text{can}(G, \alpha))$ by Lemma 54. Otherwise, we have $\text{terms}(\beta) \subseteq \text{terms}(\hat{\alpha})$ and $\beta \in \text{chase}(G \cup \{\hat{\alpha}\}, \mathcal{R})_{|\text{terms}(\hat{\alpha})}$. We then remark that $\text{expansion}(\alpha') = \text{chase}(G \cup \{\varphi^{-1}(\hat{\alpha})\}, \mathcal{R})_{|\text{terms}(\varphi^{-1}(\hat{\alpha}))}$, from which we obtain $\varphi(\beta) \in \text{expansion}(\alpha')$. (2) Suppose $\rho' \in \mathcal{R}_c$ is applicable to $G' = p_{\Phi_\alpha(G)}(\mathbf{t}) = \text{can}(G, \alpha)$ by π', and let $F' = G' \cup \{\alpha'\}$, where $\alpha' = \pi'^{\text{safe}}(\text{head}(\rho'))$. As ρ' is a complex rule, it has been created because some (G_c, α_c) belongs to $\mathcal{G}$, with $\Phi_{\alpha_c}(G_c) = \Phi_\alpha(G)$, and there is some (original) rule ρ that is applicable to G_c through some π^*, such that $\text{head}(\rho') = \text{can}(G^*, \Phi_{\alpha^*,\text{terms}(\alpha_c)}(\alpha^*))$, where $\alpha^* = \pi^{*\text{safe}}(\text{head}(\rho))$ and $G^* = \Phi_{\alpha^*,\text{terms}(\alpha_c)}(\text{chase}(G_c \cup \{\alpha^*\}, \mathcal{R})_{|\text{terms}(\alpha^*)})$. Observe that G and G_c are isomorphic, and let Ψ be an isomorphism from G_c to G. Then ρ is applicable to G via $\pi = \Psi \circ \pi^*$, and the locally complete derivation step yields $\hat{G} = \text{chase}(G \cup \{\hat{\alpha}\}, \mathcal{R})_{|\text{terms}(G \cup \{\hat{\alpha}\})}$, with $\hat{\alpha} = \pi^{\text{safe}}(\text{head}(\rho))$.

Now let φ be the (injective) extension of the identity on $\text{terms}(G)$ that maps each fresh null in α' introduced during the application of ρ' to G' to the corresponding null in $\hat{\alpha}$ introduced by the application of ρ to G. An atom β in $\text{expansion}(F')$ belongs either to $\text{expansion}(\text{can}(G, \alpha))$ or to $\text{expansion}(\alpha')$. By Lemma 54, we have $\text{expansion}(\text{can}(G, \alpha)) = G$. It follows that if $\beta \in \text{expansion}(\text{can}(G, \alpha))$, then $\varphi(\beta) = \beta \in G \subseteq \hat{G}$. Otherwise, $\beta \in \text{expansion}(\alpha')$, so $\beta \in \text{chase}(G \cup \{\varphi^{-1}(\hat{\alpha})\}, \mathcal{R})_{|\text{terms}(\varphi^{-1}(\hat{\alpha}))}$. From this we obtain $\varphi(\beta) \in \text{chase}(G \cup \{\hat{\alpha}\}, \mathcal{R})_{|\text{terms}(\hat{\alpha})} \subseteq \hat{G}$. □

Repeatedly using the first bullet point of Lemma 58, Proposition 59 proves that anything that is entailed by a locally complete $\mathcal{R}$-derivation of I is also entailed by the expansion of an $\mathcal{R}_c$-derivation of I'—the guarded translation of I.

Proposition 59. *Let $\mathcal{R}$ be a set of guarded rules and I be an instance. Let $\hat{I}_0, \ldots, \hat{I}_n$ be a locally complete $\mathcal{R}$-derivation from I. Let I' and $\mathcal{R}_c$ be defined as above. There is a classical $\mathcal{R}_c$-derivation $I' = I'_0, \ldots, I'_n$ such that for any $i \in \{1, \ldots, n\}$, there exists a homomorphism φ_i from I_i to* expansion(I'_i)*, such that for all guarded sets (G, α) with $G \subseteq I_i$, there is an atom $\beta \in I'_i$ with $\varphi_i(G) \subseteq$* expansion(β).

Proof. We prove the result by induction on the length of the locally complete derivation.

- If the locally complete derivation is of length 0, then $I_0 = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)} = \text{expansion}(I')$ by Lemma 55, and one can take the identity for φ_0. Indeed, if (G, α) is a guarded set with $G \subseteq I_0$, we let $\beta = \mathsf{can}(I^*_\alpha) \in I'$ and observe that $G \subseteq \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(\alpha)} = \text{expansion}(\beta)$.
- Otherwise, let us assume that there is a homomorphism φ_i from I_i to expansion(I'_i) such that for all guarded sets $G \subseteq I_i$, there exists $\beta \in I'_i$ such that $\varphi_i(G) \subseteq$ expansion(β). Let ρ_i be the rule applied to I_i through π_i to generate I_{i+1}, creating an atom α_{i+1}. As $\pi_i(\mathsf{body}(\rho_i))$ is a guarded set in I_i, by the induction assumption, there is $\beta_i \in I'_i$ such that $\varphi_i(\pi_i(\mathsf{body}(\rho_i))) \subseteq \text{expansion}(\beta_i)$. Let us notice that $\mathsf{chase}(\text{expansion}(\beta_i), \mathcal{R})_{|\mathsf{terms}(\text{expansion}(\beta_i))} = \text{expansion}(\beta_i)$, as it holds for any initial β_i and for any atom added by definition of complex rules. Also observe that $\beta_i = \mathsf{can}(\text{expansion}(\beta_i))$. We can thus apply Lemma 58 to infer that there exists ρ'_i applicable to β_i, whose application creates α'_{i+1}, such that there is a homomorphism from $\mathsf{chase}(\text{expansion}(\beta_i) \cup \{\alpha_{i+1}\}, \mathcal{R})_{|\mathsf{terms}(\alpha_{i+1})}$ to expansion$(\beta_i \cup \alpha'_{i+1})$ that is the identity on terms(expansion(β_i)) We define φ_{i+1} as the mapping extending φ_i by mapping a fresh null introduced in α_{i+1} to the corresponding fresh null introduced in α'_{i+1}. For any guarded set G of I_{i+1}, one of the two following case holds:
  - $\mathsf{terms}(G) \subseteq \mathsf{terms}(I_i)$, and so by induction assumption, there exists $\beta_i \in I'_i$ for which $\varphi_{i+1}(G) = \varphi_i(G) \subseteq$ expansion(β_i);
  - $\mathsf{terms}(G) \subseteq \mathsf{terms}(\alpha_{i+1})$, in which case $\alpha'_{i+1} \in I'_{i+1}$ is such that $\varphi_{i+1}(G) \subseteq \text{expansion}(\alpha'_{i+1})$.

  In particular, this implies that φ_{i+1} is a homomorphism from I_{i+1} to expansion(I'_{i+1}). □

Conversely, Proposition 60 uses the second bullet point of Lemma 58 to show that anything that is entailed by the expansion of the result of a classical $\mathcal{R}_c$-derivation of I' is also entailed by the result of a locally complete $\mathcal{R}$-derivation of I.

Proposition 60. *Let $\mathcal{R}$ be a set of guarded rules and I be an instance. Let I' and $\mathcal{R}_c$ be defined as above. For any classical $\mathcal{R}_c$-derivation $I' = I'_0, \ldots, I'_n$, there exists a locally complete $\mathcal{R}$-derivation $\hat{I}_0 = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}, \ldots, \hat{I}_n$ such that for any $i \in \{0, \ldots, n\}$, there is a homomorphism φ_i from* expansion(I'_i) *to $\hat{I}_i$.*

Proof. We show the result by induction on the length of the $\mathcal{R}_c$-derivation.

- If the $\mathcal{R}_c$-derivation is of length 0, then $I'_n = I'$, and by Lemma 55, $\text{expansion}(I') = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}$. The derivation $\hat{I}_0 = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(I)}$ is a locally complete $\mathcal{R}$-derivation of I of length 0, and one can take φ_0 as the identity;
- Let us assume that there exists a homomorphism φ_i from expansion(I'_i) to $\hat{I}_i$. Let $\rho_i \in \mathcal{R}_c$ be the rule applicable to I'_i by π_i on α that yields $I'_{i+1} = I'_i \cup \{\alpha'\}$. By Lemma 58, there exists $\rho \in \mathcal{R}$ that is applicable to expansion(α) by π' such that there is a homomorphism h from expansion$(\alpha \cup \alpha')$ to $\mathsf{chase}(H, \mathcal{R})_{|\mathsf{terms}(H)}$ with $H = \text{expansion}(\alpha) \cup \pi'^{\text{safe}}(\mathsf{head}(\rho))$ that is the identity on terms(expansion(α)). It follows in particular that ρ is applicable to $\varphi_i(\text{expansion}(\alpha)) \subseteq \hat{I}_i$ using $\pi'' = \varphi_i \circ \pi'$. We can thus let $\hat{I}_{i+1} = \mathsf{chase}(\hat{I}_i \cup \pi''^{\text{safe}}(\mathsf{head}(\rho)), \mathcal{R})_{|\mathsf{terms}(\hat{I}_i \cup \pi''^{\text{safe}}(\mathsf{head}(\rho)))}$ be the result of applying the locally

complete derivation step associated with ρ and π'' to $\hat{I}_i$. We observe that there is a homomorphism g from $\mathsf{chase}(H,\mathcal{R})_{|\mathsf{terms}(H)}$ to $\hat{I}_{i+1}$ defined by setting $g(t) = \varphi_i(t)$ for all $t \in \mathsf{terms}(\mathsf{expansion}(\alpha))$ and for all $t \in \mathsf{terms}(\pi'^{\mathsf{safe}}(\mathsf{head}(\rho))) \setminus \mathsf{terms}(\mathsf{expansion}(\alpha))$, letting $g(t)$ be the corresponding fresh term in $\pi''^{\mathsf{safe}}(\mathsf{head}(\rho))$. The desired homomorphism φ_{i+1} from $\mathsf{expansion}(I'_{i+1})$ to $\hat{I}_{i+1}$ is obtained by setting $\varphi_{i+1}(t) = \varphi_i(t)$ for all $t \in \mathsf{terms}(\mathsf{expansion}(I'_i))$ and $\varphi_{i+1}(t) = g \circ h(t)$ for all $t \in \mathsf{terms}(\mathsf{expansion}(I'_{i+1})) \setminus \mathsf{terms}(\mathsf{expansion}(I'_i)) \subseteq \mathsf{terms}(\mathsf{expansion}(\alpha')) \setminus \mathsf{terms}(\mathsf{expansion}(\alpha))$. Indeed, this is a consequence of the facts that (i) $\mathsf{expansion}(I'_{i+1}) = \mathsf{expansion}(I'_i) \cup \mathsf{expansion}(\alpha')$, (ii) φ_i is a homomorphism from $\mathsf{expansion}(I'_i)$ to $\hat{I}_i \subseteq \hat{I}_{i+1}$, (iii) $g \circ h$ is a homomorphism from $\mathsf{expansion}(\alpha \cup \alpha')$ to $\hat{I}_{i+1}$, (iv) φ_i and $g \circ h$ agree on their common domain, and (v) $\mathsf{terms}(\mathsf{expansion}(\alpha')) \cap \mathsf{terms}(\mathsf{expansion}(I'_i)) \subseteq \mathsf{terms}(\mathsf{expansion}(\alpha))$. □

The preceding proposition together with Proposition 45 directly imply the following corollary.

Corollary 61. *For any KB $(I,\mathcal{R})$ and homomorphism-closed Boolean query q on the original vocabulary, there is a locally complete $\mathcal{R}$-derivation from I resulting in $\hat{I}_m$ such that $\hat{I}_m \models q$ iff there exists a classical $(\mathcal{R}_c \cup \mathcal{R}_r)$-derivation from I' resulting in I'_n with $I'_n \models q$.*

We finally define $\mathcal{R}' = \mathcal{R}_r \cup \mathcal{R}_c$. Theorem 62, which states the correction of our reduction, is a direct consequence of Proposition 45 and Corollary 61.

Theorem 62. *Let q be a Boolean CRPQ on the original vocabulary. Then* $\mathsf{chase}(I,\mathcal{R}) \models q$ *if and only if* $\mathsf{chase}(I',\mathcal{R}') \models q$.

It remains to study the computational resources that are required to compute this reduction.

Proposition 63. *Given $(I,\mathcal{R})$, I' and $\mathcal{R}'$ as defined above can be computed in 2ExpTime in combined complexity, ExpTime in combined complexity with bounded-predicate arity, and in PTime in data complexity. The number of types (Definition 8) whose predicate appears in $\mathcal{R}'$ is at most a double exponential in I and $\mathcal{R}$ (exponential when the arity is bounded, and constant* w.r.t. *the data).*

Proof. Let w be the maximum arity of a predicate appearing in $\mathcal{R}$ or I, p be the number of such predicates, and n be the number of atoms of I, and r be the number of rules.

Let I be an instance and $\mathcal{R}$ be a set of guarded rules. We analyze the size of the instance I' and ruleset $\mathcal{R}'$:

- I' contains n atoms;
- every guarded set in $\mathcal{G}$ can be represented in single exponential size (polynomial when the arity is bounded and in data complexity): there are $p(2w)^w$ atoms that can be built using the $2w$ canonical variables, and a predicate is described by a subset of such atoms (plus the selected guard atom); the same bounds apply to space required to store a complex predicate, as each complex predicate corresponds to a guarded set;
- there is a double exponential number of complex predicates (exponential when the arity is bounded and constant in data complexity);
- for each complex predicate, there is one reconstruction rule per atom in the guarded set of the predicate, hence at most $p(2w)^w$ such rules per predicate;
- for each complex predicate, there is one complex rule per homomorphism of a rule body in $\mathcal{R}$ into the guarded set of atoms defining the predicate; in fact, it only matters where the guard atom is mapped, so we have at most $rp(2w)^w$ complex rules per new predicate;
- there is a double exponential number of types (exponential when the arity is bounded, constant in data complexity): for each complex predicate, the number of types with this predicate is equal to the number of partitions of the predicate's arguments.

To build the instance I', we first compute for each atom $\alpha \in I$ the set $\mathsf{chase}(I,\mathcal{R})_{|\mathsf{terms}(\alpha)}$. This can be done by making at most npw^w calls to a 2-ExpTime oracle for query answering under guarded rules. This is a polynomial

number of calls to an ExpTime oracle when the arity is bounded, and a polynomial number of calls to a PTime oracle in data complexity). The new instance I' can be constructed in polynomial time from the saturated atoms.

Building the set of reconstruction rules can be done in polynomial time with respect to the number of guarded sets. To build the set of complex rules, for each guarded set and each original rule ρ, we proceed as follows: apply the reconstruction rules, apply the original rule, saturate the resulting atom with respect to the original rules, and create the corresponding complex rule. The preceding operations can be performed by making an exponential number of calls to a query answering oracle for guarded rules, and thus, this step can be done in double exponential time. When the arity is bounded, this makes a polynomial number of calls to an ExpTime oracle, hence is doable in ExpTime. In data complexity, we have a constant number of calls to a PTime oracle, hence is doable in PTime. □

The next theorem follows from the provided reduction and Algorithm 2. Let us remark that the reduction to linear rules not being polynomial, we cannot directly apply Theorem 20 to obtain the desired upper bounds. However, a careful analysis of Algorithm 2 allows us to show that it actually runs polynomially in the number of types built from the predicates in $\mathcal{V}$. The proof details are provided in the appendix.

Theorem 64. *CRPQ answering under guarded rules is 2ExpTime-complete in combined complexity, ExpTime-complete in combined complexity with bounded-predicate arity, and PTime-complete in data complexity.*

We believe that the result for combined complexity in the bounded-arity case still holds for arbitrary guarded rules with multiple head atoms (translated into atomic-head rules), the key argument being again that the number of types to be considered for a fresh predicate remains polynomial. However, the proof of that claim would require updating the definitions of $\mathcal{R}_c$ and $\mathcal{R}_r$, intuitively to consider only 'relevant' guarded sets, and to revise accordingly all the subsequent statements and their proof. This would significantly complexify the arguments and obfuscate the main ideas of the reduction, hence we decided not to consider this extension.

7 Related Work

In the introduction, we gave an overview of related work on answering navigational queries with ontologies. We observed that such queries had been extensively studied within various description logic fragments, but hardly at all for existential rules. We now take a closer look at related work more specifically linked to our contributions, whether regarding the techniques used or the complexity results obtained. We recall that we use the notation (C)RPQ for *two-way* queries, whereas these are also denoted by 2(C)RPQ in other works that reserve the term (C)RPQ to the restricted form without inverse predicates (called one-way).

Concerning relevant description logics, we focus on the DL-Lite (Calvanese et al. 2007a) and $\mathcal{EL}$ (Baader et al. 2005) families, which are well-known fragments related to linear and guarded existential rules, respectively. The most studied member of the DL-Lite family is the dialect DL-Lite$_\mathcal{R}$, underpinning OWL 2 QL, whose (positive) axioms can be seen as linear rules of the form $\alpha_1 \rightarrow \alpha_2$; as usual in DLs, each α_i is a unary atom or a binary atom with distinct variables. Hence, linear rules strictly generalize DL-Lite$_\mathcal{R}$ by unrestricted predicate arity and co-occurrences of variables in an atom. In $\mathcal{EL}$, axioms can be seen as rules of the form $B[x] \rightarrow a(x)$ or $B[x] \rightarrow \exists z\ r(x,z) \wedge a(z)$ where $B[x]$ is a conjunction of unary atoms on x or a conjunction of the form $r'(x,y) \wedge a'(y)$. $\mathcal{ELH}$ adds role inclusions, i.e., rules of the form $r'(x,y) \rightarrow r(x,y)$ and $\mathcal{ELHI}$ moreover allows for inverse roles, i.e., atoms of the form $r(y,x)$. Note that $\mathcal{ELHI}$ subsumes (positive) DL-Lite$_\mathcal{R}$. Guarded rules strictly generalize $\mathcal{ELHI}$ axioms.

We mentioned in Section 3 that our algorithm for RPQ answering under linear rules (Algorithm 2) was inspired by a related algorithm for DL-Lite ontologies. More specifically, we extended the algorithm presented in (Bienvenu et al. 2015) for DL-Lite$_\mathcal{R}$ to take higher-arity predicates and co-occurrences of variables into account. Technically, the main differences are the following. In DL-Lite$_\mathcal{R}$, each detour of a path of terms to the anonymous part of the

Table 3. Data and arity-bounded combined complexities of (C)-RPQ-answering under lightweight DL ontologies

	RPQ answering		**CRPQ answering**	
Fragment	**Data**	**Combined** (b)	**Data**	**Combined** (b)
DL-Lite$_\mathcal{R}$	NL-c	PTime-c	NL-c	PSpace-c
Linear	NL-c	PTime-c	NL-c	PSpace-c
$\mathcal{ELH}$	PTime-c	PTime-c	PTime-c	PSpace-c
$\mathcal{ELHI}$	PTime-c	ExpTime-c	PTime-c	ExpTime-c
Guarded	PTime-c	ExpTime-c	PTime-c	ExpTime-c

chase goes from a constant and comes back to the same constant, i.e., for any path $p = (d_0, \ldots, d_n)$ in the chase, where only d_0 and d_n are constants, it holds that $d_0 = d_n$. This is not true anymore for linear rules. Instead, we observe that for any such path p, there is an atom $\alpha \in I$ in which d_0 and d_n both occur, and such that the chase of α contains a path isomorphic to p (see our Proposition 7). Hence, we keep track of "loops" within atoms instead of "loops" on constants. A less important extension relates to the notion of atom type, the idea being that atoms with the same type behave similarly regarding rule applications. In DL-Lite, the type of an atom is simply given by its predicate. For linear rules, we need to take co-occurrences of terms into account, hence a more general definition of type (Definition 8). Modulo these extensions, our Algorithm 2 is quite close to the algorithm for DL-Lite$_\mathcal{R}$ from (Bienvenu et al. 2015).

In Table 3 we indicate the data and combined complexities of (C)RPQ answering for the DLs mentioned above, based on (Bienvenu et al. 2015). To facilitate comparison with our results, we recall the complexities obtained for linear and guarded existential rules; we consider here combined complexity with bounded predicate arity, as DL predicates are at most binary. It is interesting to observe that the generalizations from DL-Lite$_\mathcal{R}$ to linear and from $\mathcal{ELHI}$ to guarded do not increase worst-case complexities, neither in data nor in bounded-arity combined complexity. Note however, that except for RPQ answering under linear rules (inspired by the algorithm for DL-Lite$_\mathcal{R}$), the techniques used to obtain complexity upper bounds in (Bienvenu et al. 2015) are quite different from ours, as they rely on a (non-deterministic) query rewriting algorithm that exploits the specificities of DLs.

Finally, let us mention the work in (Stefanoni et al. 2014), which studies XPath queries under the DL $\mathcal{ELRO}^+$, underpinning OWL 2 EL. This DL is incomparable with guarded existential rules, due to the presence of axioms of the form $r_1(x, x_1), \ldots, r_k(x_{k-1}, y) \rightarrow r(x, y)$. Moreover, the considered XPath queries generalize (C)RPQs by allowing to express some constraints on the nodes traversed along a path. An interesting future work would be to check whether the atom types we use in our algorithm could be extended to process those XPath queries.

Concerning existential rules, let us first point out similarities between some of our technical tools and those of previous work on CQ answering in the linear and guarded fragments. Regarding linear rules, our valid proof schemes (see Definition 27) are not far, in essence, from the proof generators from Gottlob et al. (Gottlob et al. 2015) used in a combined approach for CQ entailment under linear rules; however, the presence of transitions and their interactions with linear rules make the validity check much more intricate (see the proof of Proposition 39). Also, the reduction employed in Section 6 to lift complexity results from linear to guarded rules is very similar to that introduced in (Gottlob et al. 2023). However, as the technical lemmas provided there are not enough to be used as black boxes to get our results, we provided a novel presentation based on the notion of a locally complete derivation (Definition 41) and its properties of query entailment preservation (Proposition 45 and Corollary 61).

We now review in more detail known results about (C)RPQ answering with existential rules. Beforehand, it is worth recalling that known decidability results on CQ answering with specific existential rule classes mostly rely on properties of the *chase* or of query *rewriting*: either the universal model of the KB computed by the chase is finite or "well-shaped", which allows one to evaluate the CQ against it, or the CQ can be rewritten

using the rules into a finite query, which is then evaluated on the instance. When moving to CRPQs, it turns out that the landscape becomes quite different depending on the underlying technique. Indeed, recent work has shown that CRPQ answering is decidable for all classes of rules that guarantee the existence of a universal KB model of finite clique-width (a notion that generalizes treewidth). As pointed out in (Ostropolski-Nalewaja and Rudolph 2024), this follows from a generic result from (Feller et al. 2023). This result applies to major classes of rules with chase-based decidable CQ answering, like those defined by acyclicity notions ensuring finite chase (Grau et al. 2013) or the guarded family, which includes generalizations of the guarded class considered in this paper (Thomazo et al. 2012). This contrasts with first results on rule classes with rewriting-based decidable CQ answering, which are also provided in (Ostropolski-Nalewaja and Rudolph 2024). Indeed, RPQ answering is shown to be undecidable for first-order rewritable rule classes, also known as *fus* (Baget et al. 2011): such classes guarantee that any CQ can be rewritten as a (finite) union of CQs. This negative result already holds for *one-way* RPQs. On the positive side, RPQ answering is shown to be decidable [5] for an important fus concrete subclass, namely sticky rules (introduced in (Calì et al. 2010)). However, this decidability result does not yield any upper bound on the complexity of RPQ answering under sticky rules, and it is left as an open question whether (one-way) CRPQ answering is decidable under sticky rules.

8 Conclusion

In this paper, we provide the first complexity results for (C)RPQ answering under existential rules. These results concern two prominent classes of existential rules, namely linear and guarded rules, and distinguish between data complexity and combined complexity, considering both bounded and unbounded predicate arity. All of our complexity results are tight, thereby yielding a complete picture of the worst-case complexity of (C)RPQ answering under linear and guarded rules. Interestingly, when comparing with existing results for description logics, we observe that moving from DL-Lite$_\mathcal{R}$ to linear rules and from $\mathcal{ELHI}$ to guarded rules does not lead to any increase in data complexity, nor in bounded-arity combined complexity. Moreover, for linear rules, the data complexity for (C)RPQs is the same as for plain graph databases (namely, NL-complete).

While we now have clear picture of the complexity for guarded rules, the decidability and complexity landscapes for (C)RPQ answering for other classes of existential rules remain largely unexplored. As detailed in the related work section (Section 7), decidability has only recently been established for some major classes of rules with chase-based decidable CQ answering, including the class of frontier-guarded rules which generalizes the guarded rules considered in the present paper, but this result does not come with complexity bounds, nor a reasonably implementable algorithmic scheme. For classes with rewriting-based decidable CQ answering, there is a general undecidability result, and little is known about the decidability of concrete classes.

At present, our complexity results are purely of a theoretical nature. However, we have reason to believe that the algorithm we presented for RPQ answering under linear rules can lead to a practically efficient implementation. Indeed, the construction of the `Loop` table is data-independent, and empirical studies (Bonifati et al. 2020) have found that regular languages in real-world path queries are typically very simple (thus representable with only a handful of automata states). Moreover, to reduce computation at query time, one could perform an offline preprocessing of the data by adding all entailed binary atoms (in the spirit of combined approaches to OMQA (Kontchakov et al. 2011), which rely upon data enrichment to speed up query answering). This pre-computation of entailed facts may also enable us to devise RPQ answering procedures via rewriting to path queries over plain graph databases, thereby enabling the use of modern graph database systems (as has been explored in (Löhnert et al. 2025) for description logics). By contrast, the algorithms we devised for CRPQ answering under linear and guarded rules are not readily implementable, so new insights, possibly coupled with restrictions on query structure, will be needed to address such queries in practice.

[5] For the proof to hold, RPQs must be of the form $\Lambda(x_1, x_2)$ with x_1 and x_2 distinct variables.

Acknowledgments

This work was partially supported by the French ANR projects PAGODA (ANR-12-JS02-0007), CQFD (ANR-18-CE23-0003), and EXPAND (ANR-25-CE23-1215). We thank the reviewers for their comments that helped to enhance the quality of the paper.

References

Serge Abiteboul, Richard Hull, and Victor Vianu. 1994. *Foundations of Databases.* Addison Wesley.

Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40.

Franz Baader, Sebastian Brandt, and Carsten Lutz. 2005. Pushing the $\mathcal{EL}$ Envelope. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 364–369.

Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Swan Rocher. 2015a. Combining Existential Rules and Transitivity: Next Steps. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2720–2726.

Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Swan Rocher. 2015b. Combining Existential Rules and Transitivity: Next Steps. *CoRR* abs/1504.07443 (2015).

Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Michaël Thomazo. 2017. Answering Conjunctive Regular Path Queries over Guarded Existential Rules. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 793–799.

Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. 2011. On Rules with Existential Variables: Walking the Decidability Line. *Artif. Intell.* 175, 9-10 (2011), 1620–1654.

Catriel Beeri and Moshe Vardi. 1981. The Implication Problem for Data Dependencies. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 115. 73–85.

Catriel Beeri and Moshe Y. Vardi. 1984. A Proof Procedure for Data Dependencies. *J. ACM* 31, 4 (1984), 718–741.

Meghyn Bienvenu, Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. 2014. Nested Regular Path Queries in Description Logics. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR)*.

Meghyn Bienvenu and Magdalena Ortiz. 2015. Ontology-Mediated Query Answering with Data-Tractable Description Logics. In *Lecture Notes of the International Reasoning Web Summer School (RW) (LNCS, Vol. 9203)*. Springer, 218–307.

Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. 2015. Regular Path Queries in Lightweight Description Logics: Complexity and Algorithms. *J. Artif. Intell. Res. (JAIR)* 53 (2015), 315–374.

Meghyn Bienvenu and Michaël Thomazo. 2016. On the Complexity of Evaluating Regular Path Queries over Linear Existential Rules. In *Proceedings of the International Conference on Web Reasoning and Rule Systems (RR)*. 1–17.

Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An Analytical Study of Large SPARQL Query Logs. *VLDB J.* 29, 2-3 (2020), 655–679.

Andrea Calì, Georg Gottlob, and Michael Kifer. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.* 48 (2013), 115–174.

Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2009a. Datalog Extensions for Tractable Query Answering over Ontologies. In *Semantic Web Information Management - A Model-Based Perspective*. 249–279.

Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2009b. A General Datalog-Based Framework For Tractable Query Answering Over Ontologies. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART International Symposium on Principles of Database Systems (PODS)*. 77–86.

Andrea Calì, Georg Gottlob, and Andreas Pieris. 2010. Advanced Processing for Ontological Queries. *Proc. VLDB Endow.* 3, 1 (2010), 554–565.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007a. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. Autom. Reasoning* 39 (2007), 385–429.

Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. 2007b. Answering Regular Path Queries in Expressive Description Logics: An Automata-Theoretic Approach. In *Proceedings of the International AAAI Conference on Artificial Intelligence*. 391–396.

Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. 2009. Regular Path Queries in Expressive Description Logics with Nominals. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 714–720.

Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. 2014. Answering Regular Path Queries in Expressive Description Logics Via Alternating Tree-Automata. *Inf. Comput.* 237 (2014), 12–55.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Containment of Conjunctive Regular Path Queries with Inverse. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR)*. 176–185.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2002. Rewriting of Regular Expressions and Regular Path Queries. *J. Comput. Syst. Sci.* 64, 3 (2002), 443–465.

Ashok K. Chandra, Harry R. Lewis, and Johann A. Makowsky. 1981. Embedded Implicational Dependencies and their Inference Problem. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. 342–354.

Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. 1987. A Graphical Query Language Supporting Recursion. In *Proceedings of the ACM Special Interest Group on Management of Data Annual Conference*, Umeshwar Dayal and Irving L. Traiger (Eds.). 323–330.

Tamara Cucumides, Juan L. Reutter, and Domagoj Vrgoc. 2023. Size Bounds and Algorithms for Conjunctive Regular Path Queries. In *Proceedings of the International Conference on Database Theory (ICDT)*. 13:1–13:17.

Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. 2008. The chase revisited. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*. 149–158.

Nikola Dragovic, Cem Okulmus, and Magdalena Ortiz. 2023. Rewriting Ontology-Mediated Navigational Queries into Cypher. In *Proceedings of the International Workshop on Description Logics (DL)*.

Andrzej Ehrenfeucht and H. Paul Zeiger. 1976. Complexity Measures for Regular Expressions. *J. Comput. Syst. Sci.* 12, 2 (1976), 134–146.

Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.* 336, 1 (2005), 89–124.

Thomas Feller, Tim S. Lyon, Piotr Ostropolski-Nalewaja, and Sebastian Rudolph. 2023. Finite-Cliquewidth Sets of Existential Rules: Toward a General Criterion for Decidable yet Highly Expressive Querying. In *Proceedings of the International Conference on Database Theory (ICDT)*. 18:1–18:18.

Diego Figueira. 2021. Foundations of Graph Path Query Languages. In *Lecture Notes of the International Reasoning Web Summer School (RW) (LNCS, Vol. 13100)*. Springer, 1–21.

Daniela Florescu, Alon Levy, and Dan Suciu. 1998. Query Containment for Conjunctive Queries with Regular Expressions. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART International Symposium on Principles of Database Systems (PODS)*.

Georg Gottlob, Marco Manna, and Andreas Pieris. 2015. Polynomial Rewritings for Linear Existential Rules. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*. 2992–2998.

Georg Gottlob, Marco Manna, and Andreas Pieris. 2023. Polynomial Combined First-Order Rewritings For Linear and Guarded Existential Rules. *Artif. Intell.* 321 (2023), 103936.

Georg Gottlob and Christos H. Papadimitriou. 2003. On the Complexity of Single-Rule Datalog Queries. *Inf. Comput.* 183, 1 (2003), 104–122.

Georg Gottlob and Thomas Schwentick. 2012. Rewriting Ontological Queries into Small Nonrecursive Datalog Programs. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning*.

Gösta Grahne and Adrian Onet. 2018. Anatomy of the Chase. *Fundam. Informaticae* 157, 3 (2018), 221–270.

Bernardo Cuenca Grau, Ian Horrocks, Markus Krötzsch, Clemens Kupke, Despoina Magka, Boris Motik, and Zhe Wang. 2013. Acyclicity Notions for Existential Rules and Their Application to Query Answering in Ontologies. *J. Artif. Intell. Res. (JAIR)* 47 (2013), 741–808.

David S. Johnson and Anthony C. Klug. 1984. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *J. Comput. Syst. Sci.* 28, 1 (1984), 167–189.

Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. 2011. The Combined Approach to Ontology-Based Data Access. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*. 2656–2661.

Roman Kontchakov, Mariano Rodriguez-Muro, and Michael Zakharyaschev. 2013. Ontology-Based Data Access with Databases: A Short Course. In *Lectures Notes of the International Reasoning Web (RW) (LNCS, Vol. 8067)*. Springer, 194–229.

Egor V. Kostylev, Juan L. Reutter, and Domagoj Vrgoc. 2015. XPath for DL Ontologies. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1525–1531.

Leonid Libkin, Wim Martens, Filip Murlak, Liat Peterfreund, and Domagoj Vrgoc. 2025. Querying Graph Data: Where We Are and Where To Go. In *Companion of the the International Symposium on Principles of Database Systems (PODS)*. 9–26.

Bianca Löhnert, Nikolaus Augsten, Cem Okulmus, and Magdalena Ortiz. 2025. Towards Practicable Algorithms for Rewriting Graph Queries Beyond DL-Lite. In *Proceedings of the 22nd European Semantic Web Conference (ESWC)*. 342–361.

David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. 1979. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.* 4, 4 (1979), 455–469.

Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.* 24, 6 (1995), 1235–1258.

Marie-Laure Mugnier and Michaël Thomazo. 2014. An Introduction to Ontology-Based Query Answering with Existential Rules. In *Lectures Notes of the International Reasoning Web Summer School (RW)*. 245–278.

Magdalena Ortiz, Sebastian Rudolph, and Mantas Šimkus. 2011. Query Answering in the Horn Fragments of the Description Logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Magdalena Ortiz and Mantas Simkus. 2012. Reasoning and Query Answering in Description Logics. In *Lecture Notes of the International Reasoning Web Summer School (RW) (LNCS, Vol. 7487)*. Springer, 1–53.

Piotr Ostropolski-Nalewaja and Sebastian Rudolph. 2024. The Sticky Path to Expressive Querying: Decidability of Navigational Queries under Existential Rules. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2008. Linking Data to Ontologies. *J. Data Semant.* 10 (2008), 133–173.

Swan Rocher. 2016. *Querying Existential Rule Knowledge Bases: Decidability and Complexity. (Interrogation de Bases de Connaissances avec Règles Existentielles : Décidabilité et Complexité)*. Ph. D. Dissertation. University of Montpellier, France.

Giorgio Stefanoni, Boris Motik, Markus Krötzsch, and Sebastian Rudolph. 2014. The Complexity of Answering Conjunctive and Navigational Queries over OWL 2 EL Knowledge Bases. *J. Art. Intell. Res. (JAIR)* 51 (2014),

645–705.

Michaël Thomazo, Jean-François Baget, Marie-Laure Mugnier, and Sebastian Rudolph. 2012. A Generic Querying Algorithm for Greedy Sets of Existential Rules. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR)*.

Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. 2018. Ontology-Based Data Access: A Survey. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 5511–5519.

Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. 2019. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. *Data Intell.* 1, 3 (2019), 201–223.

A Proofs for Section 5 (CRPQ Answering under Linear Rules: Upper Bound)

Proposition 39. *Checking the validity of a proof scheme can be done in single exponential time, in polynomial space if predicate arity is bounded, and in NL if q and $\mathcal{R}$ are fixed.*

Proof. We prove here the correction of the procedure presented in the main body of the article. To simplify notations, we will use $\mathcal{CG}$ to abbreviate $\mathcal{CG}(I, \mathcal{R})$.

($\Rightarrow$) First we show that if the above procedure returns 'valid', then $\mathbb{P}$ is a valid proof scheme. Let us thus consider an execution of the above procedure that returns 'valid'. We need to show how to construct a witnessing embedding ψ that satisfies all of the validity conditions. In fact, it will prove convenient to construct instead a mapping μ from the atoms in $\mathbb{P}$ to nodes in $\mathcal{CG}$, and then let ψ be the unique mapping from $\mathsf{terms}(\mathbb{P})$ to $\mathsf{terms}(\mathsf{chase}(I, \mathcal{R}))$ that satisfies $\psi(\alpha) = \ell(\mu(\alpha))$. Note that by definition, every atom $\psi(\alpha)$ belongs to $\mathsf{chase}(I, \mathcal{R})$.

We begin by letting μ be the mapping that maps every root atom α in $\mathbb{P}$ to the (unique) root atom in $\mathcal{CG}$ satisfying $\ell(v) = \alpha$. It follows that the initial mapping ψ has as its domain all terms appearing in a root atom of $\mathbb{P}$ and maps every such term to itself. Observe that since all root atoms belong to I (else the procedure would have returned 'not valid' in Step 1), ψ trivially satisfies the first three validity conditions when restricted to the root atoms of $\mathbb{P}$. It is easily seen that for every pair α, α' of root atoms in $\mathbb{P}$, the set $\textsc{Trans}(\alpha, \alpha')$ computed in Step 1 using RPQ Entailment is exactly equal to $\mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha), \psi(\alpha'))$. Since Step 1 ensures that every $(\tau, (\alpha, \alpha')) \in \mathcal{T}$ is such that $\tau \in \textsc{Trans}(\alpha, \alpha')$, the fourth validity condition is satisfied for transitions between root atoms.

We next show how to extend μ (hence ψ) to cover the other atoms in $\mathbb{P}$. Let us take some atom α_c for which μ has not yet been defined, but whose parent α_p is in the domain of μ. We suppose that the current mapping μ induces a mapping ψ that satisfies all of the validity conditions as regards the atoms and terms in its domain (in particular, with respect to the atom α_p). We further suppose that the set $\textsc{Trans}(\alpha_p, \alpha_p)$ computed by the procedure is equal to $\mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_p))$. We know that at some point during the execution, the atom α_c was removed from Unexamined, and the inner while loop produced a sequence of atoms $\alpha_1, \ldots, \alpha_n$ such that $\alpha_1 = \alpha_p$ and $\alpha_n = \alpha_c$ (the latter holds because of the check $\alpha_c = \alpha_i$ performed after exiting the inner while loop), where $n \leq \textsc{Max} + 1$. The procedure will have also constructed for every $1 \leq j \leq n$, sets of transitions $\textsc{Trans}(\alpha_p, \alpha_j)$, $\textsc{Trans}(\alpha_j, \alpha_p)$, and $\textsc{Trans}(\alpha_j, \alpha_j)$. We will show how, for each $1 < j \leq n$, the mapping μ (and the induced mapping ψ) can be extended to the terms in α_j in such a way that the following conditions hold:

(i) $\mu(\alpha_j)$ is a child of $\mu(\alpha_{j-1})$ in $\mathcal{CG}$;
(ii) α_j and $\psi(\alpha_j)$ are isomorphic[6] w.r.t. the identity function on $\mathsf{terms}(I)$;
(iii) α_j and $\psi(\alpha_j)$ are isomorphic w.r.t. the bijection from $\mathsf{terms}(\alpha_{j-1})$ to $\mathsf{terms}(\psi(\alpha_{j-1}))$ induced by their common type;
(iv) $\textsc{Trans}(\alpha_p, \alpha_j) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_j))$;
(v) $\textsc{Trans}(\alpha_j, \alpha_p) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_j), \psi(\alpha_p))$;
(vi) $\textsc{Trans}(\alpha_j, \alpha_j) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_j), \psi(\alpha_j))$

Let us thus assume that we have already defined $\mu(\alpha_{j-1})$ in such a way as to satisfy the preceding properties, and show how to do the same for α_j. Let b be the bijection from $\mathsf{terms}(\alpha_{j-1})$ to $\mathsf{terms}(\psi(\alpha_{j-1}))$ induced by their common type. We know that α_j was obtained by applying a rule $\rho = \varphi \rightarrow \chi \in \mathcal{R}$ to α_{j-1} with some substitution σ. More precisely, $\sigma(\varphi) = \alpha_{j-1}$ and α_j is obtained from χ by replacing each frontier variable z by $\sigma(z)$ and every existential variable y by a null from $\mathsf{terms}(\alpha_c) \setminus (\mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_i))$ or a fresh null (not in $\mathsf{terms}(\mathcal{F})$) using the next available id. For convenience, we may assume that the nulls that occur in $\mathcal{F}$ or are generated by the procedure do not appear in $\mathsf{chase}(I, \mathcal{R})$. Because α_{j-1} is isomorphic to $\psi(\alpha_{j-1})$ with bijection b, we have

[6] Two atoms α_1 and α_2 are *isomorphic with respect to a bijection b* from a set of terms T_1 to a set of terms T_2 if they have the same type and, for any position i of their shared predicate, if $\alpha_1[i] \in T_1$ then $\alpha_2[i] = b(\alpha_1[i])$, otherwise $\alpha_2[i] \notin T_2$, and $\alpha_1[i]$ and $\alpha_2[i]$ are both nulls.

$b(\sigma(\varphi)) = \psi(\alpha_{j-1})$, so ρ can be applied to $\psi(\alpha_{j-1})$ under substitution $b \circ \sigma$. It follows that the atom produced by this rule application is a child of $\mu(\alpha_{j-1})$ in $\mathcal{CG}$. We let $\mu(\alpha_j)$ be this child. We observe that by construction and due to properties (ii) and (iii) holding for α_{j-1} and $\mu(\alpha_{j-1})$, the atoms α_j and $\psi(\alpha_j) = \ell(\mu(\alpha_j))$ are isomorphic with respect to b and with respect to the identity function on terms(I). Thus, properties (i)-(iii) are satisfied for α_j.

It remains to show that our definition of $\psi(\alpha_j)$ satisfies properties (iv)-(vi). For property (iv), first suppose that $\tau = (k_1, s_1, k_3, s_3) \in \textsc{Trans}(\alpha_p, \alpha_j)$. Then τ must have been added to $\textsc{Trans}(\alpha_p, \alpha_j)$ in Step 2(d) due to the presence of transitions $(k_1, s_1, k_2, s_2) \in \textsc{Trans}(\alpha_p, \alpha_{j-1})$ and $(k'_2, s_2, k_3, s_3) \in \textsc{Trans}^{\downarrow}(\alpha_j, \alpha_j)$ with $\alpha_{j-1}[k_2] = \alpha_j[k'_2]$. By assumption, we have

$$(k_1, s_1, k_2, s_2) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_{j-1}))$$

Moreover, $(k'_2, s_2, k_3, s_3) \in \textsc{Trans}^{\downarrow}(\alpha_j, \alpha_j)$ and the construction of $\textsc{Trans}^{\downarrow}(\alpha_j, \alpha_j)$ in Step 2(c) imply that the RPQ $q_\tau = \mathcal{L}_{\mathbb{A}}(s_2, s_3)(\alpha_j[k'_2], \alpha_j[k_3])$ is entailed from $(\{\alpha_j\}, \mathcal{R})$. As α_j and $\psi(\alpha_j)$ are isomorphic, it must also be the case that $\mathcal{L}_{\mathbb{A}}(s_2, s_3)(\psi(\alpha_j)[k'_2], \psi(\alpha_j)[k_3])$ is entailed from $(\{\psi(\alpha_j)\}, \mathcal{R})$. It follows that

$$(k'_2, s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\psi(\alpha_j)\},\mathcal{R})}(\psi(\alpha_j), \psi(\alpha_j))$$

Since $\psi(\alpha_j)$ appears in $\mathsf{chase}(I, \mathcal{R})$, this implies in turn

$$(k'_2, s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_j), \psi(\alpha_j))$$

From $\alpha_{j-1}[k_2] = \alpha_j[k'_2]$ and the isomorphism holding between atoms α_{j-1} and $\psi(\alpha_{j-1})$ and between atoms α_j and $\psi(\alpha_j)$, we must have $\psi(\alpha_{j-1})[k_2] = \psi(\alpha_j)[k'_2]$. We can thus combine the preceding statements to obtain

$$(k_1, s_1, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_j))$$

which establishes that

$$\textsc{Trans}(\alpha_p, \alpha_j) \subseteq \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_j)).$$

For the second inclusion in (iv), let $\tau = (k_1, s_1, k_3, s_3)$ belong to $\mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_j))$, with $\mathbb{A}$ the automaton containing states s_1, s_3. By Lemma 38, there exist

$$(k_1, s_1, k_2, s_2) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_{j-1}))$$

and

$$(k'_2, s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\psi(\alpha_j)\},\mathcal{R})}(\psi(\alpha_j), \psi(\alpha_j))$$

such that $\psi(\alpha_{j-1})[k_2] = \psi(\alpha_j)[k'_2]$. Applying our assumption, we obtain

$$(k_1, s_1, k_2, s_2) \in \textsc{Trans}(\alpha_p, \alpha_{j-1})$$

From $(k'_2, s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\psi(\alpha_j)\},\mathcal{R})}(\psi(\alpha_j), \psi(\alpha_j))$ and the fact that α_j and $\psi(\alpha_j)$ are isomorphic, we can infer $(k'_2, s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\alpha_j\},\mathcal{R})}(\alpha_j, \alpha_j)$. It follows that the RPQ $q_\tau = \mathcal{L}_{\mathbb{A}}(s_2, s_3)(\alpha_j[k'_2], \alpha_j[k_3])$ is entailed from $(\{\alpha_j\}, \mathcal{R})$, so

$$(k'_2, s_2, k_3, s_3) \in \textsc{Trans}^{\downarrow}(\alpha_j, \alpha_j)$$

because of how $\textsc{Trans}^{\downarrow}$ is constructed in Step 2(c). It follows that (k_1, s_1, k_3, s_3) was added to $\textsc{Trans}(\alpha_p, \alpha_j)$ in Step 2(d), which yields the second inclusion of property (iv). Properties (v) and (vi) can be shown analogously, utilizing Lemma 38.

Since $\alpha_n = \alpha_c$, we can conclude that the definition of $\psi(\alpha_c)$ satisfies properties (i)-(vi). It follows that the definition of $\psi(\alpha_c)$ satisfies the third validity condition (the first condition only concerns root nodes, and the second holds by definition, as explained earlier). To show the fourth condition, take some $(\tau, (\alpha_p, \alpha_c)) \in \mathcal{T}$. Since the execution succeeded, we know that $\{\tau \mid (\tau, (\alpha_p, \alpha_c)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_p, \alpha_c)$, and by property (iv), we have $\textsc{Trans}(\alpha_p, \alpha_c) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_c))$. We thus have $\tau \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_c))$. Transitions of the forms $(\tau, (\alpha_c, \alpha_p)) \in \mathcal{T}$ and $(\tau, (\alpha_c, \alpha_c)) \in \mathcal{T}$ are handled analogously, using properties (v) and (vi).

($\Leftarrow$) We suppose that $\mathbb{P} = (\mathcal{F} = (V, E), \mathcal{T})$ is a valid proof scheme w.r.t. $(I, \mathcal{R}, q)$ with witnessing embedding ψ and show that there is an execution of the above procedure that returns 'valid'.

Because of the first validity condition, we know that all root atoms in V belong to I. It follows that this check will succeed in Step 1 of the procedure. Next let (α, α') be a pair of (not necessarily distinct) root atoms in V. As argued in the first direction of the proof, the set $\textsc{Trans}(\alpha, \alpha')$ computed in Step 1 using RPQ Entailment is exactly equal to $\mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\alpha, \alpha')$ (recall that $\psi(\alpha) = \alpha$ and $\psi(\alpha') = \alpha'$, due to the first validity condition). We also know by the final validity condition that for each $(\tau, (\alpha, \alpha')) \in \mathcal{T}$, it holds that $\tau \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\alpha, \alpha')$. Combining the latter statements, we have that every $(\tau, (\alpha, \alpha')) \in \mathcal{T}$ is such that $\tau \in \textsc{Trans}(\alpha, \alpha')$. Thus, all checks in Step 1 succeed.

Let us now consider Step 2. At the start of Step 2, we initialize $\textsc{Unexamined}$ to the set of non-root nodes in V. Whenever we arrive at the beginning of the outer while loop, we choose arbitrarily some $\alpha_c \in \textsc{Unexamined}$ whose parent α_p is such that $\alpha_p \notin \textsc{Unexamined}$. A simple examination of the procedure shows that no ancestor of α_c belongs to $\textsc{Unexamined}$, hence every ancestor of α_c is either a root node (and thus was considered during Step 1) or a node in V that was selected during some iteration of the outer while loop. It follows that for every ancestor α' of α_c, the set $\textsc{Trans}(\alpha', \alpha')$ will have been computed by the procedure. We know from the previous paragraph that if α' is a root node, then $\textsc{Trans}(\alpha', \alpha')$ contains precisely the transitions in $\mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha'), \psi(\alpha'))$. We will assume that the same holds for every predecessor α' of α_c, in particular, its parent α_p, and our objective will be to show that we can perform the while loop for α_c in such a way that at the end of the inner while loop, we have $\alpha_i = \alpha_c$ and the computed sets $\textsc{Trans}(\alpha_c, \alpha_c)$, $\textsc{Trans}(\alpha_p, \alpha_c)$, and $\textsc{Trans}(\alpha_c, \alpha_p)$ satisfy:

- $\textsc{Trans}(\alpha_c, \alpha_c) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_c), \psi(\alpha_c))$
- $\textsc{Trans}(\alpha_p, \alpha_c) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_c))$
- $\textsc{Trans}(\alpha_c, \alpha_p) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_c), \psi(\alpha_p))$

Note that the preceding conditions, together with the final validity condition, imply that:

- $\{\tau \mid (\tau, (\alpha_c, \alpha_c)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_c, \alpha_c)$
- $\{\tau \mid (\tau, (\alpha_p, \alpha_c)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_p, \alpha_c)$
- $\{\tau \mid (\tau, (\alpha_c, \alpha_p)) \in \mathcal{T}\} \subseteq \textsc{Trans}(\alpha_c, \alpha_p)$

It follows that the checks for α_c at the end of the outer while loop will succeed. Since we show how to successfully perform the steps within the outer while loop for every non-root atom α_c, we will eventually reach a state in which $\textsc{Unexamined} = \emptyset$, at which point the procedure will return 'valid', as desired.

We thus let α_c be the non-root node from $\textsc{Unexamined}$ that has just been selected for examination, and let α_p be its unique parent, for which we assume that $\textsc{Trans}(\alpha_p, \alpha_p) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_p))$. The third validity condition ensures that $\psi(\alpha_c)$ can be derived from $\psi(\alpha_p)$, which ensures that there is a descendant v_n of $v_0 = \mu(\alpha_p)$ in $C\mathcal{G}$ such that $\ell(v_n) = \psi(\alpha_c)$.

We let $v_0, v_1, \ldots, v_n$ be the unique shortest path in from v_0 to v_n in $C\mathcal{G}$, and set $\beta_j = \ell(v_j)$ for every $1 \leq j \leq n$. We will use the sequence of atoms $\beta_0, \ldots, \beta_n$ in $\mathsf{chase}(I, \mathcal{R})$ to decide which rules and substitutions to guess in Step 2(a). We do not assume $n \leq \textsc{Max}$, but will instead show later that if $n > \textsc{Max}$, we can extract a subsequence of the required length with the desired properties. Thus, for the moment, our goal is to show how to successfully execute Step 2 if there were no bound on the number of iterations of the inner while loop, and afterwards we will show how the execution can be modified so as to also satisfy this bound.

Set $\gamma_0 = \alpha_p$ and denote by γ_j the atom α'_i obtained in Step 2(b) after performing j rule applications (with $1 \leq j \leq n$). We will show that for every $0 \leq j \leq n$, the following conditions hold:

(i) $\textsc{Trans}(\gamma_j, \gamma_j) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_j, \beta_j)$;
(ii) $\textsc{Trans}(\gamma_0, \gamma_j) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_j)$;
(iii) $\textsc{Trans}(\gamma_j, \gamma_0) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_j, \beta_0)$;

(iv) γ_j and β_j are isomorphic with respect to the bijection b from terms(γ_j) to terms(β_j) induced by their common type, and also w.r.t. the identity function on terms(I);
(v) $\beta_j[m] = \psi(\alpha_c)[m']$ iff $\gamma_j[m] = \alpha_c[m']$.

It follows from the last condition that $\gamma_n = \alpha_c$. We also point out that the algorithm will only keep the most recent Trans sets in memory, so in the above conditions, by $\textsc{Trans}(\gamma_j, \gamma_j)$, we mean the value of the set $\textsc{Trans}(\alpha'_i, \alpha'_i)$ at the end of the iteration in which $\alpha'_i = \gamma_j$ (and likewise for the sets $\textsc{Trans}(\gamma_0, \gamma_j)$ and $\textsc{Trans}(\gamma_j, \gamma_0)$). Further note that since $\beta_0 = \psi(\alpha_p)$, $\beta_n = \psi(\alpha_c)$, $\gamma_0 = \alpha_p$, and $\gamma_n = \alpha_c$, the above conditions imply that

- $\textsc{Trans}(\alpha_c, \alpha_c) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_c), \psi(\alpha_c))$
- $\textsc{Trans}(\alpha_p, \alpha_c) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_p), \psi(\alpha_c))$
- $\textsc{Trans}(\alpha_c, \alpha_p) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\psi(\alpha_c), \psi(\alpha_p))$

which is exactly what we need.

We first note that when $j = 0$, we have $\gamma_j = \gamma_0 = \alpha_p$, so conditions (i)-(v) hold by our earlier assumptions about α_p. Next suppose that we have already shown how to perform j iterations of the while loop (hence j rule applications), for some $0 \leq j < n$, in such a way that conditions (i)-(v) hold for γ_j. We now show how to perform the $j + 1^{\text{st}}$ iteration so that these conditions hold also for γ_{j+1}. Since v_{j+1} is a child of v_j in $\mathcal{CG}$, it follows that there exists a rule $\rho \in \mathcal{R}$ that is applicable to the atom $\beta_j = \ell(v_j)$ under a substitution σ and whose application yields the atom $\beta_{j+1} = \ell(v_{j+1})$. We also know from (iv) that γ_j and β_j are isomorphic with respect to the bijection b from terms(γ_j) to terms(β_j) induced by their common type. It follows that ρ is applicable to γ_j under the substitution mapping a variable z in the body of ρ to $b^{-1}(\sigma(z))$. We let γ_{j+1} be the atom obtained in Step 2(b) from applying ρ to γ_j under this substitution, with existential variables in the head instantiated as follows:

- If an existential variable z_0 occurs at position k in the head atom and $\beta_{j+1}[m] = \psi(\alpha_c)[m']$, then we instantiate z_0 with $\alpha_c[m']$.
- All other existential variables in the rule head are instantiated with fresh nulls (not in terms($\mathcal{F}$) nor already used earlier in the procedure) using the next available ids.

Observe that the preceding instantiation is well defined as whenever an existential variable z_0 is instantiated with $\beta_{j+1}[m] = \psi(\alpha_c)[m']$, it must be the case that $\beta_{j+1}[m] = \psi(\alpha_c)[m']$ is a null value that was created in β_{j+1} in $\mathcal{CG}$. In particular, this means that $\alpha_c[m']$ is a null that does not occur in γ_j nor $\gamma_0 = \alpha_p$ (indeed, if $\alpha_c[m']$ were to occur in γ_j, then by property (v), $\psi(\alpha_c)[m']$ would appear in β_j). We further observe that condition (iv) holds: γ_{j+1} is isomorphic to β_{j+1} under the bijection from terms(γ_{j+1}) to terms(β_{j+1}) induced by their common type, as well as w.r.t. the identity function on terms(I) (for the latter, note that any constant from terms(I) that occurs in γ_{j+1} must have appeared in γ_j, and hence in the same positions in β_j, and vice-versa). Moreover, by construction, condition (v) is also satisfied: by induction for values from α_c or $\psi(\alpha_c)$ that appear already in γ_j or β_j, and by the policy for null naming for freshly instantiated nulls.

Now that we have the new atom γ_{j+1}, we will consider in Step 2(c) each possible transition $\tau = (k_1, s_1, k_2, s_2)$ w.r.t. q and $(\gamma_{j+1}, \gamma_{j+1})$ and construct the corresponding RPQ $q_\tau = \mathcal{L}_{\mathbb{A}}(s_1, s_2)(\gamma_{j+1}[k_1], \gamma_{j+1}[k_2])$ (with $\mathbb{A}$ the automaton containing states s_1, s_2). We will then call an RPQ Entailment oracle to decide whether q_τ is entailed from $(\{\gamma_{j+1}\}, \mathcal{R})$, and if q_τ is entailed, we will store τ in $\textsc{Trans}^{\downarrow}(\gamma_{j+1}, \gamma_{j+1})$. We note that since γ_{j+1} and β_{j+1} are isomorphic, q_τ is entailed from $(\{\gamma_{j+1}\}, \mathcal{R})$ iff $q'_\tau = \mathcal{L}_{\mathbb{A}}(s_1, s_2)(\beta_{j+1}[k_1], \beta_{j+1}[k_2])$ is entailed from $(\{\beta_{j+1}\}, \mathcal{R})$. The latter implies that $(k_1, s_1, k_2, s_2) \in \mathcal{T}_{q,\mathsf{chase}(\{\beta_{j+1}\},\mathcal{R})}(\beta_{j+1}, \beta_{j+1})$. Putting this together, we have that $\textsc{Trans}^{\downarrow}(\gamma_{j+1}, \gamma_{j+1}) = \mathcal{T}_{q,\mathsf{chase}(\{\beta_{j+1}\},\mathcal{R})}(\beta_{j+1}, \beta_{j+1})$.

We now compute in (d), (e), and (f) the sets $\textsc{Trans}(\gamma_{j+1}, \gamma_{j+1})$, $\textsc{Trans}(\gamma_0, \gamma_{j+1})$, and $\textsc{Trans}(\gamma_{j+1}, \gamma_0)$, by using the transitions present in $\textsc{Trans}^{\downarrow}(\gamma_{j+1}, \gamma_{j+1})$, $\textsc{Trans}(\gamma_0, \gamma_j)$, $\textsc{Trans}(\gamma_j, \gamma_0)$, and $\textsc{Trans}(\gamma_j, \gamma_j)$. We know from our induction hypothesis that conditions (i)-(iii) hold, meaning that: $\textsc{Trans}(\gamma_j, \gamma_j) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_j, \beta_j)$,

$\textsc{Trans}(\gamma_0, \gamma_j) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_j)$, and $\textsc{Trans}(\gamma_j, \gamma_0) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_j, \beta_0)$. It follows from Lemma 38 that conditions (i)-(iii) hold also for γ_{j+1}. Indeed, to obtain (ii), we need to show that $\textsc{Trans}(\gamma_0, \gamma_{j+1}) = \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_{j+1})$. First suppose $(k_1, s_1, k_3, s_3) \in \textsc{Trans}(\gamma_0, \gamma_{j+1})$. Then because of the construction in (d), we know that there exist transitions $(k_1, s_1, k_2, s_2) \in \textsc{Trans}(\gamma_0, \gamma_j)$ and $(k_2', s_2, k_3, s_3) \in \textsc{Trans}^{\downarrow}(\gamma_{j+1}, \gamma_{j+1})$ such that $\gamma_j[k_2] = \gamma_{j+1}[k_2']$. Using the earlier equalities, we get $(k_1, s_1, k_2, s_2) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_j)$ and $(k_2', s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\beta_{j+1}\},\mathcal{R})}(\beta_{j+1}, \beta_{j+1})$, which yields $(k_1, s_1, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_{j+1})$ by the first item of Lemma 38. For the other direction, suppose $(k_1, s_1, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_{j+1})$. Then by the first item of Lemma 38, there must exist $(k_1, s_1, k_2, s_2) \in \mathcal{T}_{q,\mathsf{chase}(I,\mathcal{R})}(\beta_0, \beta_j)$ and $(k_2', s_2, k_3, s_3) \in \mathcal{T}_{q,\mathsf{chase}(\{\beta_{j+1}\},\mathcal{R})}(\beta_{j+1}, \beta_{j+1})$ with $\beta_j[k_2] = \beta_{j+1}[k_2']$. Using the earlier equalities, we get $(k_1, s_1, k_2, s_2) \in \textsc{Trans}(\gamma_0, \gamma_j)$ and $(k_2', s_2, k_3, s_3) \in \textsc{Trans}^{\downarrow}(\gamma_{j+1}, \gamma_{j+1})$, which implies that the transition (k_1, s_1, k_3, s_3) was added to $\textsc{Trans}(\gamma_0, \gamma_{j+1})$ during Step 2(d). The arguments for conditions (i) and (iii) proceed similarly.

To complete the proof, we must argue that we can make the described execution respect the bound $\textsc{Max}$ on the number of iterations of the inner while loop. Consider some non-root node α_c and its parent α_p, and let $\beta_0, \ldots, \beta_n$ (with $\beta_0 = \alpha_p$ and $\beta_n = \alpha_c$) be the sequence of atoms in $\mathsf{chase}(I, \mathcal{R})$ that were used to decide which rules and substitutions to guess in Step 2(a). We let $\rho_1, \ldots, \rho_n$ and $\sigma_1, \ldots, \sigma_n$ be the chosen sequences of rules and substitutions. If $n \leq \textsc{Max}$, then the execution of the inner while loop we defined for α_c already respects the bound. Let us thus suppose that $n > \textsc{Max}$. Thus, since $n > \textsc{Max}$, there must exist $0 \leq j < k \leq n$ such that:

(a) α_j and α_k are isomorphic w.r.t. the identity on $\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)$
(b) $\textsc{Trans}(\alpha_j, \alpha_j) = \textsc{Trans}(\alpha_k, \alpha_k)$
(c) $\textsc{Trans}(\alpha_p, \alpha_j) = \textsc{Trans}(\alpha_p, \alpha_k)$
(d) $\textsc{Trans}(\alpha_j, \alpha_p) = \textsc{Trans}(\alpha_k, \alpha_p)$

Indeed, there can be at most $P * (|A| + |\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)|)^A$ ways of selecting a type and deciding which terms from $\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)$ appear in which positions, and there are $N^2 * A^2$ possible transitions between any pair of atoms, hence at most $2^{N^2 * A^2}$ possible sets of transitions (we recall that P and A are respectively the number of predicates and maximum predicate arity for the ruleset $\mathcal{R}$, and N is the total number of states across all of the automata in q). Intuitively, conditions (a)-(d) ensure that α_j and α_k behave the same, so the idea is to jump straight from α_j to α_{k+1}, and in this manner obtain a shorter sequence that still defines a successful execution of the inner while loop but now respects the $\textsc{Max}$ bound. Of course, since α_j and α_k may use different values for terms not in $\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)$, we need to do some renaming of nulls to obtain the new shorter sequence. So, let π_k be the bijection from $\mathsf{terms}(\alpha_k)$ to $\mathsf{terms}(\alpha_j)$ witnessing the isomorphism from item (a). Then we can apply the rule ρ_k to α_j using the substitution σ_k' that maps t to $\pi_k(\sigma_k(t))$. Call α_{k+1}' the result of this rule application. It follows from item (a) that α_{k+1} and α_{k+1}' are isomorphic w.r.t. the identity on $\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)$ (we let π_{k+1} be a witnessing bijection from α_{k+1} to α_{k+1}'). Moreover, by items (b)-(d) and the way $\textsc{Trans}$ is constructed in the procedure, we must also have:

- $\textsc{Trans}(\alpha_{k+1}, \alpha_{k+1}) = \textsc{Trans}(\alpha_{k+1}', \alpha_{k+1}')$
- $\textsc{Trans}(\alpha_p, \alpha_{k+1}) = \textsc{Trans}(\alpha_p, \alpha_{k+1}')$
- $\textsc{Trans}(\alpha_{k+1}, \alpha_p) = \textsc{Trans}(\alpha_{k+1}', \alpha_p)$.

Thus, α_{k+1} and α_{k+1}' behave the same in all relevant respects, and in particular, we can apply rule ρ_{k+1} to α_{k+1}' using the substitution σ_{k+1}' defining by setting $\sigma_{k+1}'(t) = \pi_{k+1}(\sigma_{k+1}(t))$. It is readily verified, by repeating the preceding argument, that α_{k+1} and α_{k+1}' are isomorphic w.r.t. the identity on $\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)$. Continuing in this manner, we can define a sequence $\alpha_{k+1}', \ldots, \alpha_n'$ such that for every $k+1 \leq u < n$:

- α_{u+1} and α_{u+1}' are isomorphic w.r.t. the identity on $\mathsf{terms}(I) \cup \mathsf{terms}(\alpha_p) \cup \mathsf{terms}(\alpha_c)$, as witnessed by a bijection π_u from α_{u+1} to α_{u+1}'
- α_{u+1}' is obtained from α_u' by applying ρ_u using substitution σ_u' defined by $\sigma_u'(t) = \pi_u(\sigma_u(t))$

- $\textsc{Trans}(\alpha_{u+1}, \alpha_{u+1}) = \textsc{Trans}(\alpha'_{u+1}, \alpha'_{u+1})$
- $\textsc{Trans}(\alpha_p, \alpha_{u+1}) = \textsc{Trans}(\alpha_p, \alpha'_{u+1})$
- $\textsc{Trans}(\alpha_{u+1}, \alpha_p) = \textsc{Trans}(\alpha'_{u+1}, \alpha_p)$.

It follows that by guessing the sequence of rules $\rho_1, \ldots, \rho_{j-1}, \rho_k, \rho_{k+1}, \ldots, \rho_n$ and sequence of substitutions $\sigma_1, \ldots, \sigma_{j-1}, \sigma'_k, \sigma'_{k+1}, \ldots, \sigma'_n$, we are able to correctly perform the inner while loop. Moreover, by iterating this shortening operation, we will eventually obtain sequences of rules and substitutions which allow us to terminate within Max steps while satisfying the other three conditions at the end of Step 2.

We have thus shown that it is possible to define for every valid proof scheme w.r.t. $(I, \mathcal{R}, q)$ an execution of the procedure that returns 'valid', which completes the proof. □

B Proofs for Section 6 (CRPQ Answering under Guarded Rules: Upper Bound)

Proposition 48. *Let $\mathcal{R}$ be a set of guarded rules and I be an instance. For each atom $\alpha \in I$, we note $I^*_\alpha = \mathsf{chase}(I, \mathcal{R})_{|\mathsf{terms}(\alpha)}$. Then:*

(⇒) *For any $\mathcal{R}$-derivation from I to I_n, we can define for every $\alpha \in I$ an $\mathcal{R}$-derivation from I^*_α to I'_α such that $\bigcup_{\alpha \in I} I'_\alpha \models I_n$;*

(⇐) *Let for any $\alpha \in I$ an $\mathcal{R}$-derivation from I^*_α to I'_α; there exists an $\mathcal{R}$-derivation from I to I' such that $I' \models \bigcup_{\alpha \in I} I'_\alpha$.*

Proof. (⇐) Consider an $\mathcal{R}$-derivation D from some I^*_α to I'_α. There is an $\mathcal{R}$-derivation D_1 from $\{\alpha\}$ to some I^1_α such that $I^*_\alpha \subseteq I^1_\alpha$, *i.e.*, $I^1_\alpha = I^*_\alpha \cup X$. Let us now build a derivation D_2 from I^1_α, applying rules using the same homomorphisms (unless already applied) in the same order as in D. The result of this derivation is $I'_\alpha \cup X$. The derivation obtained from the concatenation of D_1 and D_2 is a derivation from $\{\alpha\}$ (and by consequence I) to $I' = I'_\alpha \cup X$ (and by consequence $I' = I'_\alpha \cup X \cup F$), and thus $I' \models I'_\alpha$.

We conclude by pointing out that the concatenation of all the derivations as build previously from I, for all atoms $\alpha \in I$ result in some I' containing $\bigcup_{\alpha \in I} I'_\alpha$, and thus $I' \models \bigcup_{\alpha \in I} I'_\alpha$. (⇒) We show the converse entailment by induction on the length of a derivation associated with an initial portion of $\mathsf{chase}(I, \mathcal{R})$: given any derivation D^i of length $i \geq 0$ from I, we build derivations D^i_α (of length at most i) from I^*_α for each $\alpha \in I$, such that $\bigcup_{\alpha \in I} \mathsf{atoms}(D^i_\alpha) \models \mathsf{atoms}(D^i)$.

More precisely, we prove the following: For all $i \geq 0$, for all derivation $D^i = I_0 (= I), \ldots, I_i$, there are h_i, ψ_i, and derivations D^i_α for each $\alpha \in I$ such that:

(1) D^i_α is a derivation from I^*_α (of length less or equal to i)
(2) h_i is a homomorphism from I_i to $\bigcup_{\alpha \in I} \mathsf{atoms}(D^i_\alpha)$;
(3) ψ_i is a function that assigns to each atom $\beta \in I_i$ an atom $\alpha \in I$ such that for any atom $\beta' \in I_i$ with $\mathsf{terms}(\beta') \subseteq \mathsf{terms}(\beta), h_i(\beta') \in \mathsf{atoms}(D^i_\alpha)$.

Note that the homomorphism h_i (Condition 2) shows that $\bigcup_{\alpha \in F} \mathsf{atoms}(D^i_\alpha) \models \mathsf{atoms}(D^i)$.

For $i = 0$, we define each D^0_α as the empty derivation from I^*_α, h_0 as the identity on terms of I and ψ_0 as the identity on atoms of I. These choices fulfill our induction assumption, because:

(1) Empty derivations are derivations;
(2) As D^0_α are empty derivations, it holds that $\mathsf{atoms}(D^0_\alpha) = I^*_\alpha$, hence h_0 is indeed a homomorphism from I to $\bigcup_{\alpha \in I} \mathsf{atoms}(D^i_\alpha)$;
(3) Let $\beta' \in I$ such that $\mathsf{terms}(\beta') \subseteq \mathsf{terms}(\beta)$. Then $\beta' \in I^*_\beta = I^*_{\psi_0(\beta)}$.

For the induction step, let us assume that $\{D^i_\alpha\}_{\alpha \in I}$, h_i, ψ_i are defined and satisfy the three conditions, and let π_{i+1} and ρ_{i+1} be such that I_{i+1} is obtained from I_i by applying rule ρ_{i+1} using π_{i+1}. Let g_{i+1} be the guard of ρ_{i+1}. As $\pi_{i+1}(g_{i+1})$ belongs to I_i, $\psi_i(\pi_{i+1}(g_{i+1}))$ is defined. Let us denote this atom by α_{r_i}. Let α_{i+1} be the atom created by the application of ρ_{i+1} under π_{i+1}. We assume w.l.o.g that α_{i+1} does not belong to I_i.

We define $\{D_\alpha^{i+1}\}_{\alpha \in I}$, h_{i+1}, and ψ_{i+1} as follows:

- ψ_{i+1} is the extension of ψ_i defined by $\psi_{i+1}(\alpha_{i+1}) = \alpha_{r_i}$;
- $D_\alpha^{i+1} = D_\alpha^i$ if $\alpha \neq \alpha_{r_i}$. For α_{r_i}, we consider the application of ρ_{i+1} to $\mathsf{atoms}(D^i_{\alpha_{r_i}})$ under $h_i \circ \pi_{i+1}$. Note that for each atom $b \in \mathsf{body}(\rho_{i+1})$, it holds that $h_i(\pi_{i+1}(b)) \in \mathsf{atoms}(D_{\alpha_{r_i}})$, as $\mathsf{terms}(b) \subseteq \mathsf{terms}(g_{i+1})$, hence $\mathsf{terms}(\pi_{i+1}(b)) \subseteq \mathsf{terms}(\pi_{i+1}(g_{i+1}))$, and, since $\psi_i(\pi_{i+1}(g_{i+1})) = \alpha_{r_i}$, by Condition 3, we get $h_i(\pi_{i+1}(b)) \in \mathsf{atoms}(D_{\alpha_{r_i}})$. Therefore, $h_i \circ \pi_{i+1}$ defines a homomorphism from $\mathsf{body}(\rho_{i+1})$ to $\mathsf{atoms}(D^i_{\alpha_{r_i}})$. We denote by β_{i+1} the atom created by this rule application. If $\beta_{i+1} \in \mathsf{atoms}(D^i_{\alpha_{r_i}})$ we set $D^{i+1}_{\alpha_{r_i}} = D^i_{\alpha_{r_i}}$, otherwise $D^{i+1}_{\alpha_{r_i}}$ is the extension of $D^i_{\alpha_{r_i}}$ by the application of ρ_{i+1} under $h_i \circ \pi_{i+1}$.
- For every null e occurring in α_{i+1}, there is a null e' occurring at the same position of β_{i+1}, and we extend h_i to h_{i+1} by setting $h_{i+1}(e) = e'$.

We now check that the built objects still fulfill the induction statement:

(1) By the induction assumption, for all $\alpha \in I$, D^i_α is a derivation from I^*_α of length less or equal to i. Hence, for all $\alpha \neq \alpha_{r_i}$, $D^{i+1}_\alpha = D^i_\alpha$ is a derivation from I^*_α (of length less of equal to i), and, for α_{r_i}, $D^{i+1}_{\alpha_{r_i}}$ is the extension of $D^i_{\alpha_{r_i}}$ by a rule application, hence of length less or equal to $i+1$;

(2) α_{i+1} is the only new atom in I_{i+1}. By construction of h_{i+1} and $D^{i+1}_{\alpha_{r_i}}$, $h_{i+1}(\alpha_{i+1})$ belongs to $\mathsf{atoms}(D^{i+1}_{\alpha_{r_i}})$, hence h_{i+1} is a homomorphism from I_{i+1} to $\bigcup_{\alpha \in I} \mathsf{atoms}(D^{i+1}_\alpha)$;

(3) We check that, for all atoms β_1 and β_2 in I_{i+1} with $\mathsf{terms}(\beta_2) \subseteq \mathsf{terms}(\beta_1)$, $h_{i+1}(\beta_2) \in \mathsf{atoms}(D^{i+1}_{\psi_{i+1}(\beta_1)})$. This holds for $\beta_1 = \beta_2 = \alpha_{i+1}$, since by construction, $h_{i+1}(\alpha_{i+1}) \in \mathsf{atoms}(D^{i+1}_{\alpha_{r_i}})$ with $\alpha_{r_i} = \psi_{i+1}(\alpha_{i+1})$. Let us consider the atoms $\beta' \in I_{i+1}$ with $\beta' \neq \alpha_{i+1}$ such that (1) $\mathsf{terms}(\beta') \subseteq \mathsf{terms}(\alpha_{i+1})$ or (2) $\mathsf{terms}(\alpha_{i+1}) \subseteq \mathsf{terms}(\beta')$. For case (1), since β' was already in I_i, $\mathsf{terms}(\beta') \subseteq \mathsf{terms}(\pi_{i+1}(g_{i+1}))$, and by induction assumption $h_{i+1}(\beta') \in \mathsf{atoms}(D_{\psi_{i+1}(\pi_{i+1}(g_{i+1}))}) = \mathsf{atoms}(D_{\psi_{i+1}(\alpha_{i+1})})$. For case (2), either $\mathsf{terms}(\alpha_{i+1}) \subseteq \mathsf{terms}(I)$, then $\alpha_{i+1} \in I^*_{\psi_{i+1}(\beta')} \subseteq \mathsf{atoms}(D^{i+1}_{\psi_{i+1}(\beta')})$, or α_{i+1} contains at least one null. Since $\mathsf{terms}(\alpha_{i+1}) \subseteq \mathsf{terms}(\beta')$, this null was already present in $\pi_{i+1}(g_{i+1})$. By the induction assumption, $\psi_i(\pi_{i+1}(g_{i+1})) = \psi_i(\beta')$, hence $\psi_{i+1}(\beta') = \alpha_{r_i}$. By construction of h_{i+1} and $D^{i+1}_{\alpha_{r_i}}$, $h_{i+1}(\alpha_{i+1}) \in \mathsf{atoms}(D^{i+1}_{\alpha_{r_i}})$. □

Theorem 64. *CRPQ answering under guarded rules is 2ExpTime-complete in combined complexity, ExpTime-complete in combined complexity with bounded-predicate arity, and PTime-complete in data complexity.*

Proof. The lower bounds come from CQ answering under guarded rules. For the upper bound, we use the reduction of CRPQ answering over guarded rules to CRPQ answering over linear rules, that is formalized in Theorem 62. We observe that building the set of rules $\mathcal{R}'$ takes double exponential time, so simply running the algorithm for linear rules over the constructed KB $(I', \mathcal{R}')$ does not yield a procedure running in 2ExpTime for guarded rules. Let us revisit the complexity of the algorithm for linear rules, starting with the RPQ answering procedure from Section 3:

- for RPQ answering under linear rules, the first step is the creation of a Loop table, which can be done in time polynomial with respect to the number of types. Combining this observation with the bounds on the number of types from Proposition 63, we can see that this step will run in double exponential time w.r.t. the input (resp. exponential time when the arity is bounded, polynomial time in data complexity).
- once the Loop table has been computed, it remains to apply Algorithm 2:
  - the number of iterations of the while loop is independent of the rules, and polynomial in the number of terms of the instance, which is unchanged by our transformation;
  - at each iteration of the while loop, we guess a constant from the instance, a state from the NFA representing the RPQ, and either a transition in the NFA or a type. There are polynomially many constants, states, and transitions. It follows that the overall number of different possible guesses at a given iteration is double

exponential in general case (exponential for the bounded-arity case, polynomial for data complexity), and the guess can be stored in exponential space (polynomial space for bounded-arity or data complexity);
  - once the preceding guesses have been made, the algorithm either checks whether the guessed type occurs in the Loop table, or performs an entailment check;
  - checking that a given type belong to the correct cell of the Loop table can be done in polynomial time with respect to the number of types (double exponential in general, single exponential when the arity is bounded, polynomial in data complexity);
  - entailment checking can be performed by a non-deterministic procedure whose space is polynomially bounded w.r.t. the maximum size of a type, which is single exponential in the size of the original input (polynomial when the arity is bounded, constant in data complexity).

Putting the above together, we obtain that RPQ answering under the set of linear rules that is output by our reduction can be done in double exponential time with respect to the initial parameters (and in ExpTime when arity is bounded, and in PTime in data complexity).

We next consider the procedure for CRPQ answering:

- the size (*i.e.*, the number of nodes) of the largest proof scheme to be considered is independent of both rules and data;
- the validity of a proof scheme can be checked as in the proof of Proposition 39:
  - each root belonging to I' : as I' has already been computed, this check can be performed in exponential time, polynomial in data complexity and with bounded arity, as the predicate names can be of exponential size with respect to the arity;
  - computing the set of transitions: doable in double exponential time, exponential time when the arity is bounded, polynomial in data complexity, as this is can be done through a quadratic number of calls to RPQ answering under the modified linear ruleset;
  - selection of atoms, rules: doable in exponential time, polynomial in data complexity and when the arity is bounded, as predicate names can be of exponential size in the arity;
  - rule application: exponential time, polynomial in data complexity and when the arity is bounded, same reason
  - the while loop of **Step 2** of the query answering algorithm is iterated at most Max times, which is a simple exponential (polynomial in data complexity) as per the proof of Proposition 39. □

Received 23 July 2025; Accepted 17 June 2026.