

Polynomial combined first-order rewriting for query answering over non-recursive Datalog knowledge bases

Nicola Leone¹, Marco Manna¹, Aldo Ricioppo¹ and Lucio Pio Savonarola Gentile¹

¹AI Lab, Department of Mathematics and Computer Science, University of Calabria, Via Pietro Bucci, 87036 Rende (CS), Italy

Abstract

Ontology-mediated query answering, extensively studied over the past two decades, consists in evaluating a conjunctive query (CQ) over a knowledge base (KB) formed by a relational database (DB) and an ontology. To exploit existing RDBMSs, the pure first-order (FO) rewriting approach compiles the ontology and the CQ into a single FO query (or a single UCQ, as a commonly considered special case) to be evaluated over the DB. However, it is not always applicable, and when it is, it may yield an FO rewriting of exponential size. A more refined approach is the so-called combined FO rewriting, which additionally compiles the DB and the ontology into a transformed DB; it applies beyond the limits of pure rewriting and often avoids such blow-up. Among ontology languages of broad interest, nrDatalog is the well-established class of positive, non-recursive sets of Datalog rules. While pure UCQ rewritings for this language do, in general, yield queries of unavoidable exponential size, it remains open whether this blow-up can be avoided by targeting full FO queries through pure or combined rewritings. We take a first step towards addressing this gap by proposing a polynomial-time combined FO rewriting for nrDatalog KBs. First, we compute a normal form of the input enforcing uniform predicate arities, absence of constants and joins in rule heads, and a strict layered structure in which each rule derives head atoms using only predicates from the previous level in its body. Second, from the normalized input, we construct a transformed DB and an FO query consisting essentially of a conjunction of nested implications combining existential and universal quantification with conjunctions and disjunctions of equalities between terms, whose innermost consequent is an atom over a database predicate and whose nesting depth matches that of the ontology. Importantly, unlike existing applications of the combined approach, no ontology reasoning is performed in either phase, as the final rewriting is obtained through lightweight syntactic transformations of the input.

Keywords

Ontology-mediated query answering, Query rewriting, non-recursive Datalog, Knowledge representation, Combined rewriting, Conjunctive queries

1. Introduction

1.1. Context

Ontologies provide an explicit specification of the conceptualization of an application domain and constitute a fundamental component of modern knowledge-based systems. Among the ontology languages proposed in the literature, Description Logics (DLs) [1] and Datalog-based formalisms, including Datalog itself [2, 3], existential rules [4, 5], Datalog[±] [6, 7, 8], and Datalog[∃] [9, 10], are arguably the most prominent. In this setting, a knowledge base (KB) consists of a relational database (DB), representing extensional knowledge, together with an ontology, representing intensional knowledge. A central reasoning task is *ontology-mediated query answering* (OMQA) [11, 12, 13, 14], also known as ontology-based data access (OBDA) [15, 16, 17, 18] and ontology-based query answering (OBQA) [19, 20, 21, 22]. In all these settings, a conjunctive query (CQ) is evaluated over a DB while taking into account the additional knowledge entailed by the ontology.

A widely accepted principle is that scalable query answering should exploit the optimization capabilities of existing relational database management systems (RDBMSs). This observation has motivated extensive research on first-order (FO) rewriting techniques over the last two decades. The classical, or *pure*, FO rewriting approach leaves the input DB unchanged and compiles the ontology together with the CQ into a single FO query that can be directly evaluated by an RDBMS [23, 24, 25, 26, 27].

Datalog 2.0 2026: 6th International Workshop on the Resurgence of Datalog in Academia and Industry

✉ leone@unical.it (N. Leone); marco.manna@unical.it (M. Manna); aldo.ricioppo@unical.it (A. Ricioppo); piolucio99@outlook.it (L. P. Savonarola Gentile)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

As a special case, the FO fragment of unions of CQs (UCQs) is commonly adopted as the target query language since a pure UCQ rewriting can often be obtained whenever a pure FO rewriting exists [28]. This strategy is highly successful for lightweight ontology languages such as the DL-Lite family, but it is inherently limited. On the one hand, many ontology languages of practical interest, including EL-based ontologies, Datalog, and guarded Datalog[±], do not admit pure FO rewritings. On the other hand, even when pure rewritings exist, the rewritten query may be exponentially larger than the original CQ and ontology, and this blow-up is unavoidable even for simple ontology languages such as OWL 2 QL.

A more refined approach is the so-called *combined* FO rewriting [29, 30, 31, 32], in which, in addition to compiling the ontology and the query into an FO query, the database and the ontology are compiled into a transformed database. The *schema*—namely, the finite set of predicate symbols occurring in the DB, the ontology, and the query, together with their arities—is given as input to both rewriting modules. If the transformed DB can be computed efficiently, the evaluation of the resulting FO query can still effectively exploit standard RDBMS technology, similarly to the pure approach. Moreover, combined rewritings naturally apply to ontology languages that do not admit pure FO rewritings and frequently yield polynomial-time constructions. Polynomial combined FO rewritings have been established for several Datalog-based ontology languages, possibly under additional structural assumptions on the underlying schema or on the ontology [32, 33, 34, 35].

1.2. Motivation and objectives

Among ontology languages, nrDatalog [3] constitutes a well-established class of positive sets of Datalog rules. Its defining feature is that rules induce a dependency graph over predicates, where there is an arc from a predicate P to a predicate Q if the latter occurs in the head of a rule whose body contains P . The non-recursive condition requires this graph to be acyclic. Such a property makes nrDatalog expressive enough for several OMQA scenarios. Indeed, nrDatalog has been even used as a target rewriting language alternative to FO queries [26, 36, 37].

Pure UCQ rewritings for nrDatalog yield, in general, queries of unavoidable exponential size. For example, consider the family $\{(\Sigma_n, q_n(x_0, x_n))\}_{n>0}$ of ontology-mediated queries, where $q_n(x_0, x_n)$ is the CQ $\exists x_1 \cdots \exists x_{n-1} A_1(x_0, x_1), \dots, A_n(x_{n-1}, x_n)$ and Σ_n is the nrDatalog ontology consisting of the rules $R_i(x, y) \rightarrow A_i(x, y)$ and $S_i(x, y) \rightarrow A_i(x, y)$, for each $i \in \{1, \dots, n\}$. For this family, every pure UCQ rewriting has at least exponential size in n . Polynomial-size pure FO rewritings, however, do exist for this family:

$$Q_n(x_0, x_n) = \exists x_1 \cdots \exists x_{n-1} \bigwedge_{i \in \{1, \dots, n\}} (A_i(x_{i-1}, x_i) \vee R_i(x_{i-1}, x_i) \vee S_i(x_{i-1}, x_i)).$$

However, it remains open whether the exponential blow-up can be avoided for arbitrary nrDatalog KBs by targeting full FO queries, via either pure or combined rewritings. Indeed, consider the family $\{(\Sigma'_n, q'_n(x_0, x_n))\}_{n>0}$, where $q'_n(x, z) = A_n(x, z)$ and Σ'_n is the nrDatalog ontology consisting of the rules: $R(x, y) \rightarrow A_0(x, y)$; $S(x, y) \rightarrow A_0(x, y)$; and $A_{i-1}(x, y), A_{i-1}(y, z) \rightarrow A_i(x, z)$ for each $i \in \{1, \dots, n\}$. For this family, every pure UCQ rewriting has at least double-exponential size in n . By applying the same strategy used to construct $Q_n(x_0, x_n)$ above, a pure FO rewriting of single-exponential size can be obtained. But can one do better?

In Proof Theory, a common way to abbreviate a proof is to introduce a fresh predicate P as a placeholder for an FO formula $\varphi(\vec{x})$. More precisely, one introduces the sentence $\forall \vec{x} (P(\vec{x}) \leftrightarrow \varphi(\vec{x}))$ in the proof, usually referred to as a *definition*, and uses the atom $P(\vec{x})$ in place of the formula $\varphi(\vec{x})$ wherever the latter occurs in the proof. A naive elimination of the definition consists in expanding the abbreviation, i.e., replacing every occurrence of $P(\vec{x})$ in the proof with $\varphi(\vec{x})$. This may, however, cause an exponential blow-up in the length of the proof. Pudlák [38] and Avigad [39] showed that definitions can nevertheless be eliminated with only polynomial overhead. Since definitions bear a close resemblance to Datalog rules, it is natural to ask whether these techniques can be directly transferred to our setting. There are, however, some differences. First, their approach is designed for definitions in which all fresh predicates have the same arity. Second, fresh predicates are “purely intentional”; in

contrast, a predicate occurring in a Datalog rule-head may also occur in the database, and is therefore both extensional and intensional. Third, unlike definitions, Datalog rule-heads may contain constants and repeated variables. Nonetheless, the fact that definitions can be eliminated with only polynomial overhead provides a useful indication that a similar phenomenon may hold for nrDatalog rules as well.

Our objective is therefore to study OMQA over nrDatalog KBs and to devise a simple, self-contained, and polynomial-time computable FO rewriting, at least in the combined setting.¹

1.3. Contribution

The main contribution of the work is a polynomial-time combined FO rewriting for OMQA over nrDatalog KBs, obtained through a two-phase construction that proceeds as follows:

1. First, we compute a normal form of the input that preserves query answers and enforces uniform predicate arities, a strict layered structure in which each predicate is assigned to a unique level and rules derive predicates only from the previous level, as well as the absence of constants and joins in rule heads.
2. Second, starting from the normalized input, we construct a transformed database together with an equivalent FO query through further syntactic transformations. The resulting FO query essentially consists of a conjunction of nested implications whose nesting depth matches that of the ontology dependency graph. The implications combine standard FO constructs, namely existential and universal quantification, conjunction, disjunction, and equality, yielding formulas reminiscent of those introduced by Fischer and Rabin for Presburger arithmetic [40] and of the quantified Boolean formulas used in the classical PSPACE-hardness proof for TrueQBF [41, Theorem 8.9].

Overall, the construction establishes polynomial combined FO rewritability for nrDatalog KBs, extending FO rewriting techniques to a fragment for which only exponential pure rewritings were previously known. Interestingly, unlike typical applications of the combined approach, the transformation of the DB requires no ontology reasoning. It essentially amounts to a slightly different encoding of the input facts, in which we have a single predicate and some fresh constants, whose number grows linearly with both the size of the ontology and the number of original predicates. Hence, although this transformation is formally required to obtain the desired FO rewriting, from an implementation perspective the database can be maintained directly in the encoded representation, with only negligible additional information being introduced when needed.

2. Preliminaries

For the entire paper, we fix three pairwise disjoint countably infinite sets of symbols: P of *predicates*, C of *constants*, and V of *variables*. A (*relational*) *schema*, typically denoted by S , is any finite subset of P . When referring to variables, we typically employ symbols such as x, y, z , and variations thereof. Constants and variables are collectively referred to as *terms*. Each predicate P has an associated *arity*, denoted by $\text{ar}(P)$. An *atom* is an expression of the form $P(\bar{t})$, where P is a predicate and $\bar{t} = t_1, \dots, t_{\text{ar}(P)}$ are terms. If $\text{ar}(P) = 0$, then $P()$ is a propositional atom, also written P , where parentheses are omitted as usual. An atom is *ground* if it contains no variables. Consider two sets A and B of atoms. A *C-homomorphism* from A to B is a function h from the terms occurring in A to the terms occurring in B such that (i) $h(c) = c$ for each $c \in C$; and (ii) for each atom $\alpha = P(t_1, \dots, t_{\text{ar}(P)})$ in A , the atom $h(\alpha) = P(h(t_1), \dots, h(t_{\text{ar}(P)}))$ belongs to B .

¹Note that materializing in the rewritten DB all the consequences of the ontology (i.e., the whole minimal model), without modifying the query, would trivially yield a combined rewriting. However, this rewriting is not of polynomial-size, since, for unbounded arities, the minimal model can in general be exponential in size.

2.1. Datalog knowledge bases

A *database* is a finite set of ground atoms. By $\text{const}(D)$ we denote the set of all constants occurring in D . A *Datalog rule* is a first-order sentence σ of the form:

$$\forall x_1 \cdots \forall x_m (B_1(\bar{t}_1) \wedge \cdots \wedge B_k(\bar{t}_k) \rightarrow H(\bar{t}_0)),$$

where $H, B_1, \dots, B_k$ are predicates with $k \geq 1$, each $\bar{t}_i$ is a sequence of terms from $\mathcal{C} \cup \{x_1, \dots, x_m\}$ with $m \geq 0$, and each variable occurring in $\bar{t}_0$ also occurs in some $\bar{t}_i$. Universal quantifiers are usually omitted, and conjunctions of atoms are often written using commas instead of $\wedge$. The atom $\text{head}(\sigma) = H(\bar{t}_0)$ is called the *head* of σ , while the set $\text{body}(\sigma) = \{B_1(\bar{t}_1), \dots, B_k(\bar{t}_k)\}$ is called the *body* of σ . A variable in $\bar{t}_0$ is called a *frontier* variable; accordingly, a variable occurring in $\text{body}(\sigma)$ but not in $\text{head}(\sigma)$ is called *non-frontier*. A *Datalog ontology* is a finite set of Datalog rules. Let us denote by **Datalog** the class of all Datalog ontologies. A *Datalog knowledge base* (Datalog KB) is a pair $K = (D, \Sigma)$, where D is a database and Σ is a Datalog ontology.

A database D satisfies a Datalog rule σ if, for each homomorphism h from $\text{body}(\sigma)$ to D , it holds that $h(\text{head}(\sigma)) \in D$. A database D satisfies a Datalog ontology Σ if D satisfies each rule in Σ . Consider a Datalog KB $K = (D, \Sigma)$. In general, D does not satisfy Σ . We thus consider extensions of D that satisfy Σ . A *model* of K is any database $M \supseteq D$ such that M satisfies Σ . We write $\text{mods}(K)$ for the set of all the models of K . A model M is *minimal* if there exists no other model M' of K such that $M' \subset M$. The following is well-known.

Proposition 1 ([3]). *Every Datalog KB admits at least one model and exactly one minimal model.*

Consider a Datalog ontology Σ . Its associated *dependency graph* is the directed graph $\mathcal{G}(\Sigma) = (N, A)$, whose set of nodes N contains all predicates occurring in Σ , and where there is an arc $(p, r) \in A$ whenever p occurs in the body of a rule whose head predicate is r . We say that Σ is *non-recursive* if its dependency graph $\mathcal{G}(\Sigma)$ is a directed acyclic graph. The class of all non-recursive Datalog ontologies is denoted by **nrDatalog**. A *non-recursive Datalog KB* is a Datalog KB $K = (D, \Sigma)$, where $\Sigma \in \text{nrDatalog}$.

2.2. Conjunctive queries

A *conjunctive query* (CQ) is a first-order expression $q(\bar{x})$ of the form

$$\exists y_1 \cdots \exists y_k (P_1(\bar{u}_1) \wedge \cdots \wedge P_n(\bar{u}_n)),$$

where $P_1, \dots, P_n$ with $n \geq 1$ are predicates, $\bar{x} = x_1, \dots, x_m$ with $m \geq 0$ is the list of all the free variables of $q(\bar{x})$ (possibly with repetitions and in some order), and each $\bar{u}_i$ is a sequence of terms from $\mathcal{C} \cup \{x_1, \dots, x_m\} \cup \{y_1, \dots, y_k\}$ with $k \geq 0$. Let $\text{atm}(q)$ denote the set of atoms occurring in $q(\bar{x})$. The *output* of $q(\bar{x})$ on a database D is the set $\text{out}(q, D)$ of all tuples $(c_1, \dots, c_m)$ of constants such that there exists a $\mathcal{C}$ -homomorphism h from $\text{atm}(q)$ to D with $h(x_i) = c_i$ for each $i \in \{1, \dots, m\}$.

2.3. Query answering over Datalog knowledge bases

Consider a Datalog KB $K = (D, \Sigma)$ and a CQ $q(\bar{x})$. The (*certain*) *answers* to q over K are defined as

$$\text{ans}(q, D, \Sigma) = \bigcap_{M \in \text{mods}(K)} \text{out}(q, M).$$

To compare query answering instances, especially when they are defined over different relational schemas, it is convenient to capture the query, the database, and the ontology together as a single object called a *reasoning triple*. In what follows, given a reasoning triple $T = (q, D, \Sigma)$ as above, we shall write $\text{ans}(T)$ as a shorthand for $\text{ans}(q, D, \Sigma)$. If S is the set of all predicates occurring in T , we say that T is over the schema S . We can now define equivalence between reasoning triples.

Definition 1. Two reasoning triples T_1 and T_2 , possibly over different schemas, are said to be *equivalent*, written $T_1 \equiv T_2$, if $\text{ans}(T_1) = \text{ans}(T_2)$.

For Datalog KBs, the computation of certain answers can be characterized in terms of minimal models. Indeed, by Proposition 1, every Datalog KB K admits a unique minimal model, denoted by M_K . Since conjunctive queries are monotone and M_K is contained in every model of K , evaluating a CQ over M_K yields the certain answers over K . The next well-known proposition makes this observation precise.

Proposition 2 ([3]). *Consider a Datalog KB $K = (D, \Sigma)$ and a CQ $q(\bar{x})$. If M_K is the minimal model of K , then $\text{ans}(q, D, \Sigma) = \text{out}(q, M_K)$.*

2.4. First-order queries

The queries produced by our rewriting are not conjunctive but arbitrary first-order queries, evaluated over a single database. We recall their semantics, adapted from [3].

A *first-order query* $Q(\bar{x})$ is a first-order formula over a schema S , built from atoms and equalities $t = t'$ by means of $\neg, \wedge, \vee, \exists, \forall$, with free variables $\bar{x}$. By $\text{const}(Q)$ we denote the set of constants appearing in Q . Let η be an *assignment* sending each variable of Q to a constant. Satisfaction of Q in D under η , written $(D, \eta) \models Q$, is defined inductively:

- $(D, \eta) \models x=y$ iff $\eta(x) = \eta(y)$, and $(D, \eta) \models x=a$ iff $\eta(x) = a$;
- $(D, \eta) \models R(u_1, \dots, u_k)$ iff $R(\eta(u_1), \dots, \eta(u_k)) \in D$;
- $\wedge, \vee, \neg$ are read classically;
- $(D, \eta) \models \exists x \psi$ (resp. $\forall x \psi$) iff $(D, \eta[x/a]) \models \psi$ for some (resp. every) $a \in \text{const}(D) \cup \text{const}(\psi)$.

When Q is a sentence, D either satisfies Q or not; in the first case D is a *model* of Q . The *output* of $Q(\bar{x})$ over D is $\text{out}(Q, D) = \{\bar{a} \in (\text{const}(D) \cup \text{const}(Q))^{\lvert \bar{x} \rvert} : (D, \eta) \models Q \text{ for } \eta(\bar{x}) = \bar{a}\}$, the tuples of constants making Q true. Finally, every conjunctive query is a first-order query, and for CQs the two notions of output agree: $\text{out}(q, D)$ defined via C-homomorphisms coincides with the one just given. The certain-answer semantics of Section 2.3 instead applies to conjunctive queries only, resting on a monotonicity that general first-order queries lack.

3. The Core Principle of the Combined Rewriting

The combined rewriting we are going to develop compiles a CQ together with an ontology into a FO query to be evaluated over a (transformed) database. The obstacle we must overcome is that the obvious way of doing this, namely unfolding the query through the rules until only database predicates remain, duplicates predicates uncontrollably: a single predicate may end up occurring an exponential number of times, and the rewritten query blows up accordingly. The whole construction rests on one simple idea to contain this duplication. Let us explain this idea with the aid of a simple example.

Consider an nrDatalog ontology consisting of two rules deriving the predicate T from P , namely $\sigma_1 : P(x, y), P(y, z) \rightarrow T(x, z)$ and $\sigma_2 : P(x, z) \rightarrow T(x, z)$, which together read “ $T(x, z)$ holds whenever there is a path of length two from x to z , or a direct edge”. Answering the CQ $q(x, z) = T(x, z)$ over some database D in which we know a priori the predicate T does not appear, amounts to unfolding T through its rules, yielding a (pure) rewritten UCQ of the form

$$Q(x, z) = (\exists y P(x, y) \wedge P(y, z)) \vee P(x, z).$$

Here, the predicate P is now interrogated not merely several times, but in several different *ways*, one per rule; and the variable y of σ_1 , its *non-frontier* variable, absent from the head, lives only inside its own alternative. To collapse every occurrence of P into one, the formula must be able to *name* the rule it is currently simulating: its guard has to say “if the firing rule is σ_1 , the required P -tuples are these; if

it is σ_2 , it is that one". We therefore attach to each rule σ a fresh constant c_σ serving as its name, and let an existential variable $rule$ choose which rule fires. Hence, we can write the following FO query:

$$Q'(x, z) = \exists rule \exists y \left[\text{RuleId}(rule) \wedge \forall w_1 \forall w_2 (\Phi \rightarrow P(w_1, w_2)) \right],$$

where the guard's antecedent in the implication is:

$$\Phi = (rule = c_{\sigma_1} \wedge ((w_1 = x \wedge w_2 = y) \vee (w_1 = y \wedge w_2 = z))) \vee (rule = c_{\sigma_2} \wedge (w_1 = x \wedge w_2 = z)).$$

A single occurrence of P now serves both rules: the value guessed for $rule$ activates one disjunct of Φ , and the universal guard demands exactly the P -tuples that rule requires. Notice that $rule$ plays two distinct roles. Inside Φ it is matched against c_{σ_1} or c_{σ_2} to *select* which P -tuples are required; but it also occurs in the separate atom $\text{RuleId}(rule)$, which insists that $rule$ actually *be* one of the two rule names, provided that the rewritten database D' extends D with the atoms $\text{RuleId}(c_{\sigma_1})$ and $\text{RuleId}(c_{\sigma_2})$. This is what keeps the construction sound: were $\text{RuleId}(rule)$ dropped, one could satisfy $\exists rule$ with any other constant appearing in the transformed DB at hand or in Q' , making both disjuncts of Φ false for every w_1, w_2 , the implication *vacuously* true, and Q' return *every* pair (x, z) . This addition is in line with the combined rewriting approach, since it does not depend on the original CQ q and introduces only one constant per rule. Hence, D' differs from D by only a number of facts linear in the size of the ontology. The rules thereby cease to be part of the program and become ordinary data that the query inspects, a single occurrence of P , guarded by facts now living in the database.

Let us analyze the behavior of the rewritten FO query Q' on the rewritten database $D' = D \cup \{\text{RuleId}(c_{\sigma_1}), \text{RuleId}(c_{\sigma_2})\}$, where $D = \{P(1, 2), P(2, 3), P(1, 4)\}$. According to the rewriting, the set of constants used to interpret Q' is $\{1, 2, 3, 4, c_{\sigma_1}, c_{\sigma_2}\}$. First, it is easy to see that $ans(q, D, \Sigma) = \{(1, 2), (2, 3), (1, 3), (1, 4)\}$. To see why this set coincides with $out(Q', D')$, observe that: for $(1, 3)$ one has to guess $rule = c_{\sigma_1}$ and $y = 2$, so that it asks for $P(1, 2)$ and $P(2, 3)$. For the remaining tree pairs $(1, 2)$, $(2, 3)$, and $(1, 4)$ one has to guess $rule = c_{\sigma_2}$. All other pairs are instead rejected, as no choice of rule and witness meets the guard.

4. Rule Normalization

Before transforming an nrDatalog reasoning triple T into an equivalent first-order pair of polynomial size, we normalize its structure through three hierarchical *triple normal forms* (TNFs), each refining the previous one:

- 1TNF ensures uniform arity of predicates;
- 2TNF enforces, in addition, a strict layered structure of the rules;
- 3TNF also forbids variable joins and constants in rule heads.

Accordingly, we design three polynomial-time operators: *pad* maps any triple to 1TNF, *layerize* takes 1TNF to 2TNF, and *anonymize* takes 2TNF to 3TNF. Their successive application yields a fully normalized triple:

$$T^* = \text{anonymize}(\text{layerize}(\text{pad}(T))).$$

This triple is in 3TNF and serves as input to the final first-order rewriting phase.

4.1. Padding

We begin with the first structural requirement.

Definition 2. A reasoning triple is in 1TNF if all predicates occurring in the triple have the same arity, and such arity is strictly greater than zero. ■

Our goal is to design a uniform rewriter *pad* that transforms any reasoning triple $T = (q, D, \Sigma)$ over a schema S into an equivalent 1TNF triple in polynomial time. Let $\chi = \max\{\text{ar}(P) \mid P \in S\}$. If $\chi > 0$, then the rewriting $\text{pad}(T) = (q', D', \Sigma')$ is defined as follows. Each atom of the form $P(t_1, \dots, t_{\text{ar}(P)})$ is transformed by replacing P with a fresh predicate P' of arity χ whenever $\text{ar}(P) < \chi$, and by extending its argument list with a fixed constant $c \in \mathbb{C}$, obtaining:

$$P'(t_1, \dots, t_{\text{ar}(P)}, c^{\chi - \text{ar}(P)}),$$

where $c^{\chi - \text{ar}(P)}$ denotes the sequence $c, \dots, c$ of length $\chi - \text{ar}(P)$. Atoms whose predicates already have arity χ are left unchanged. Differently, if $\chi = 0$, namely when all predicates are propositional, then every atom $P()$ is replaced by an atom $P'(c)$, where P' is a fresh unary predicate and $c \in \mathbb{C}$ is the same fixed constant used for padding in the case $\chi > 0$.

Intuitively, *pad* embeds lower-arity predicates into a common positive arity, preserving their content and distinguishing them via fresh symbols; propositional predicates become unary through a single dummy argument. The choice of c is arbitrary.

Example 1. Consider the following reasoning triple $T = (q, D, \Sigma)$, with maximum predicate arity 3: $q(x) = \exists y (R(x, y) \wedge S(y))$, $D = \{P(a, b, c), R(c, b), R(d, a)\}$, $\Sigma = \{P(x, z, y) \wedge R(y, z) \rightarrow S(x)\}$. The minimal model of (D, Σ) is $M = D \cup \{S(a)\}$. Indeed, there exists a $\mathbb{C}$ -homomorphism h from the body of the rule to M such that $h(P(x, z, y)) = P(a, b, c)$ and $h(R(y, z)) = R(c, b)$. Hence, $h(S(x)) = S(a)$ must belong to M . Since this is the case, and since no other $\mathbb{C}$ -homomorphism from the rule body to M exists, M is the minimal model. Therefore, evaluating q over M yields $\text{ans}(T) = \{(d)\}$.

After applying the *pad* operator, we obtain the equivalent 1TNF triple $T' = (q', D', \Sigma')$ where: $q'(x) = \exists y (R'(x, y, c) \wedge S'(y, c, c))$, $D' = \{P(a, b, c), R'(c, b, c), R'(d, a, c)\}$, and finally $\Sigma' = \{P(x, z, y) \wedge R'(y, z, c) \rightarrow S'(x, c, c)\}$. By the same argument, the minimal model of (D', Σ') is $M' = D' \cup \{S'(a, c, c)\}$, whence $\text{ans}(T') = \text{ans}(T) = \{(d)\}$. ■

The padding operator uniformizes arities without altering database semantics or rule satisfaction, so any evaluation over the original minimal model is mirrored in the padded one. Being a purely syntactic expansion, it increases the size of the triple by a factor bounded by χ .

Proposition 3. Consider a reasoning triple $T = (q, D, \Sigma)$, where $\Sigma \in \text{nrDatalog}$. Then $T \equiv \text{pad}(T)$.

4.2. Layerization

We proceed with the second requirement of our normalization. Consider a DAG $G = (N, A)$. The *level* of a node $v \in N$, denoted by $\text{lev}(v, G)$, is the length of the longest path in G ending in v . Formally, if v has in-degree 0, then $\text{lev}(v, G) = 0$; otherwise, $\text{lev}(v, G) = 1 + \max\{\text{lev}(u, G) \mid (u, v) \in A\}$. The *level* of G , denoted by $\text{lev}(G)$, is $\max\{\text{lev}(u, G) \mid u \in N\}$. In particular, if $A = \{(u_0, u_1), (u_1, u_2), \dots, (u_{\ell-1}, u_\ell)\}$ (with $i \neq j$ if, and only if, $u_i \neq u_j$), then we say that G is a *chain* of the form $\langle u_0, \dots, u_\ell \rangle$.

A reasoning triple $T = (q, D, \Sigma)$ obeys a chain G of the form $\langle L_0, \dots, L_\ell \rangle$ if

- (i) $\mathcal{G}(\Sigma) = G$;
- (ii) each atom in D is over the predicate L_0 ;
- (iii) each atom in q is over the predicate L_ℓ ;
- (iv) Σ contains a *propagation* rule of the form $L_{i-1}(\bar{y}) \rightarrow L_i(\bar{y})$, for each $i \in \{1, \dots, \ell\}$.

Intuitively, (i) makes the dependency graph a chain, (ii) and (iii) constrain the interface so that all facts are over L_0 and all query atoms over L_ℓ , and (iv) links consecutive layers. Together they induce a unique layered schema $\{L_0, \dots, L_\ell\}$ and force every derivation to progress along the chain, without shortcuts.

Definition 3. A reasoning triple is in 2TNF if it is in 1TNF, it obeys a chain, and the arity of its underlying schema is at least 2. ■

Our goal is to design a uniform rewriter *layerize* that takes as input any reasoning triple $T = (q, D, \Sigma)$ in 1TNF and transforms it, in polynomial time, into an equivalent 2TNF triple that obeys a chain of the form $\langle L_0, \dots, L_\ell \rangle$, where $\ell = \text{lev}(\mathcal{G}(\Sigma))$.

Specifically, the rewriting $\text{layerize}(T) = (q', D', \Sigma')$ is defined as follows. For each predicate P occurring in T , let $c_P \in C$ be a fresh constant naming P , and let $L_0, \dots, L_\ell$ be fresh predicates of arity $\chi + 1$, where χ is the common arity of the predicates of T . The database D' is obtained from D by replacing each atom of the form $P(t_1, \dots, t_{\text{ar}(P)})$ with $L_0(c_P, t_1, \dots, t_{\text{ar}(P)})$. Concerning the ontology, Σ' contains the propagation rules: $L_0(\bar{y}) \rightarrow L_1(\bar{y}); \dots; L_{\ell-1}(\bar{y}) \rightarrow L_\ell(\bar{y})$ where $\bar{y} = y_0, \dots, y_\chi$. In addition, for each rule $\sigma \in \Sigma$ of the form $B_1(\bar{t}_1), \dots, B_k(\bar{t}_k) \rightarrow H(\bar{t}_0)$ with $\text{lev}(H, \mathcal{G}(\Sigma)) = i$, the ontology Σ' contains the rewritten rule:

$$L_{i-1}(c_{B_1}, \bar{t}_1), \dots, L_{i-1}(c_{B_k}, \bar{t}_k) \rightarrow L_i(c_H, \bar{t}_0)$$

Intuitively, the construction assigns to each predicate a copy at every level of the layering and enforces that whenever a rule has a predicate R in the head at level i , then all its body predicates are taken from level $i - 1$. Finally, the query q' is obtained from q by replacing each atom of the form $P(t_1, \dots, t_\chi)$ with $L_\ell(c_P, t_1, \dots, t_\chi)$.

Example 2. Consider the database $D = \{A(1, 2), A(2, 3), D(3, 4)\}$, the ontology $\Sigma = \{A(x, y) \rightarrow B(x, y); B(x, y), A(y, z) \rightarrow C(x, z)\}$, and the query $q(x, z) = \exists y C(x, y), D(y, z)$. In the dependency graph $\mathcal{G}(\Sigma)$, the predicates A and D have in-degree 0, so $\text{lev}(A, \mathcal{G}(\Sigma)) = 0$ and $\text{lev}(D, \mathcal{G}(\Sigma)) = 0$. Since A triggers B in the first rule, we have $\text{lev}(B, \mathcal{G}(\Sigma)) = 1$. Finally, for the second rule, C depends on both B and A , yielding $\text{lev}(C, \mathcal{G}(\Sigma)) = 1 + \max\{\text{lev}(B, \mathcal{G}(\Sigma)), \text{lev}(A, \mathcal{G}(\Sigma))\} = 2$. Thus, the maximum level is $\ell = 2$. By applying the layerization, we get: $D' = \{L_0(c_A, 1, 2), L_0(c_A, 2, 3), L_0(c_D, 3, 4)\}$. Moreover, Σ' contains the propagation rules: $L_0(y_0, y_1, y_2) \rightarrow L_1(y_0, y_1, y_2); L_1(y_0, y_1, y_2) \rightarrow L_2(y_0, y_1, y_2)$; along with the rewritten ontological rules aligned to their respective head levels: $L_0(c_A, x, y) \rightarrow L_1(c_B, x, y); L_1(c_B, x, y), L_1(c_A, y, z) \rightarrow L_2(c_C, x, z)$. The rewritten query targets the top layer $\ell = 2$: $q'(x, z) = \exists y L_2(c_C, x, y), L_2(c_D, y, z)$. From the first rule of Σ , we derive $B(1, 2)$ and $B(2, 3)$; from the second one, combining $B(1, 2)$ with $A(2, 3)$, we derive $C(1, 3)$. Accordingly, $\text{ans}(q, D, \Sigma) = \{(1, 4)\}$. The layerized triple reproduces this behavior: the propagation rules lift every L_0 -atom to L_1 , providing $L_1(c_A, 2, 3)$; the first rewritten rule yields $L_1(c_B, 1, 2)$ and $L_1(c_B, 2, 3)$; the second combines $L_1(c_B, 1, 2)$ with the propagated $L_1(c_A, 2, 3)$ to derive $L_2(c_C, 1, 3)$; one further propagation lifts $L_1(c_D, 3, 4)$ to level 2, so that q' returns $(1, 4)$. Note the double role of propagation: it makes the level-0 predicate A available to a rule whose head lives at level 2, and carries D , untouched by any rule, up to the top layer. Hence $\text{ans}(\text{layerize}(T)) = \text{ans}(T) = \{(1, 4)\}$. ■

The layerization operator reifies predicate names as constants and redistributes derivations along the levels: each rewritten rule draws its premises from the level right below its head, while the propagation rules make every derived atom persist upward, so that no rule application is lost. Hence an atom $P(\bar{a})$ belongs to the minimal model of the input triple if, and only if, $L_\ell(c_P, \bar{a})$ belongs to that of the rewritten one; since the rewritten query interrogates the top layer, the answers coincide. The transformation is purely syntactic and increases the size of the triple by at most a polynomial factor.

Proposition 4. Consider a triple $T = (q, D, \Sigma)$ in 1TNF with $\Sigma \in \text{nrDatalog}$. Then, $T \equiv \text{layerize}(T)$.

4.3. Anonymization

We conclude with the third structural requirement.

Definition 4. A reasoning triple is in 3TNF if it is in 2TNF and the head of each rule contains neither constants nor repeated variables.

Our goal is to design a uniform rewriter *anonymize* that takes as input a 2TNF reasoning triple $T = (q(\bar{x}), D, \Sigma)$ obeying a chain $\langle L_0, \dots, L_\ell \rangle$, with $\ell = \text{lev}(\mathcal{G}(\Sigma))$, and computes in polynomial time

an equivalent 3TNF reasoning triple obeying a chain $\langle P_0, \dots, P_\ell \rangle$. In particular, let χ denote the arity of each predicate L_i ; then each predicate P_i has arity $\chi + 1$.

Specifically, $\text{anonymize}(T) = (q', D', \Sigma')$ is defined as follows. Let $\star$ and $\sharp$ denote two fixed distinct constants. For each atom of the form $L_0(c_1, \dots, c_\chi)$ of D we add to D' the atom $P_0(\star, c_1, \dots, c_\chi)$. Moreover, let Δ collect all the constants occurring in D and Σ ; for each constant $c \in \Delta \cup \{\star\}$ we add to D' the atom $P_0(\sharp, c, \dots, c)$. Finally, the query q' is obtained from q by replacing each atom $L_\ell(t_1, \dots, t_\chi)$ with $P_\ell(\star, t_1, \dots, t_\chi)$. Intuitively, $\star$ -prefixed atoms mimic the facts of D , whereas $\sharp$ -prefixed ones supply constants to be propagated into rule heads or enforce joins among distinct head variables. These auxiliary atoms use the same predicates P_i , so the layered architecture required by 2TNF is preserved. The propagation rules of Σ (required by 2TNF) are translated into the corresponding propagation rules over the new predicates: for each $i \in \{1, \dots, \ell\}$, the ontology Σ' contains the rule $P_{i-1}(x_0, x_1, \dots, x_\chi) \rightarrow P_i(x_0, x_1, \dots, x_\chi)$. These rules copy *all* atoms upward, regardless of their flag: in particular, they make the $\sharp$ -atoms available at every level, where the auxiliary atoms of the rewritten rules expect to find them. Accordingly, every rule $\rho \in \Sigma$, rather than a propagation, of the form: $L_{i-1}(\bar{t}_1), \dots, L_{i-1}(\bar{t}_k) \rightarrow L_i(\bar{t}_0)$, with $1 \leq i \leq \ell$, is rewritten into a rule $\rho' \in \Sigma'$ defined as:

$$P_{i-1}(\star, \bar{t}_1), \dots, P_{i-1}(\star, \bar{t}_k), P_{i-1}(\sharp, x_0, \star, \dots, \star), \Phi_\sharp \rightarrow P_i(x_0, x_1, \dots, x_\chi)$$

where both P_{i-1} and P_i have arity $\chi + 1$. Moreover, $x_0, x_1, \dots, x_\chi$ is a sequence of pairwise distinct, fresh variables, and $\Phi_\sharp$ is an auxiliary conjunction of atoms over P_{i-1} . The parts of ρ' are constructed according to the following directives:

1. For each $j \in \{1, \dots, \chi\}$, if $\bar{t}_0[j]$ is some constant $c \in \mathbf{C}$, then append to the conjunction $\Phi_\sharp$ the atom $P_{i-1}(\sharp, x_j, c, \dots, c)$.
2. For each $j \in \{1, \dots, \chi\}$, if $\bar{t}_0[j]$ is some variable $z \in \mathbf{V}$ and there exists some $\gamma < j$ being the leftmost position where z occurs in $\bar{t}_0$, then append to $\Phi_\sharp$ the atom $P_{i-1}(\sharp, x_j, x_\gamma, \dots, x_\gamma)$.
3. For each $\delta \in \{1, \dots, k\}$ and $j \in \{1, \dots, \chi\}$, if $\bar{t}_\delta[j]$ is a variable not occurring in $\bar{t}_0$ or a constant, then $\bar{\tau}_\delta[j] = \bar{t}_\delta[j]$; else, $\bar{\tau}_\delta[j] = x_p$, where p is the smallest position where $\bar{t}_\delta[j]$ occurs in $\bar{t}_0$.

Example 3. To illustrate this third step, consider a 2TNF reasoning triple $T = (q, D, \Sigma)$ obeying the chain $\langle L_0, L_1, L_2 \rangle$, where $\Delta = \{a, b, c, d\}$ and $\chi = 3$: $q(x) = \exists y L_2(x, c, y)$, $D = \{L_0(a, b, c), L_0(c, a, d)\}$,

$$\Sigma = \{ L_0(\bar{y}) \rightarrow L_1(\bar{y}); \quad L_1(z, y, c) \wedge L_1(c, z, d) \rightarrow L_2(z, c, z); \quad L_1(\bar{y}) \rightarrow L_2(\bar{y}) \}.$$

Besides the propagated copies of the facts of D at levels 1 and 2, the minimal model of (D, Σ) contains the atom $L_2(a, c, a)$, derived by the intermediate rule from the copies $L_1(a, b, c)$ and $L_1(c, a, d)$. Hence, $\text{ans}(T) = \{(a)\}$. The rewritten 3TNF triple $\text{anonymize}(T) = (q', D', \Sigma')$ operates on predicates P_0, P_1, P_2 of arity 4: $q'(x) = \exists y P_2(\star, x, c, y)$,

$$D' = \{P_0(\star, a, b, c), P_0(\star, c, a, d)\} \cup \{P_0(\sharp, e, e, e) \mid e \in \{a, b, c, d, \star\}\},$$

$$\Sigma' = \left\{ \begin{array}{l} P_0(x_0, x_1, x_2, x_3) \rightarrow P_1(x_0, x_1, x_2, x_3); \\ P_1(\star, x_1, y, c) \wedge P_1(\star, c, x_1, d) \wedge P_1(\sharp, x_0, \star, \star) \wedge \\ \quad P_1(\sharp, x_2, c, c) \wedge P_1(\sharp, x_3, x_1, x_1) \rightarrow P_2(x_0, x_1, x_2, x_3); \\ P_1(x_0, x_1, x_2, x_3) \rightarrow P_2(x_0, x_1, x_2, x_3) \end{array} \right\}.$$

The transformation shifts the structural constraints from the head into the body, leaving the new head free of constants and joins: the fresh variables $x_0, \dots, x_3$ may map to any term, while the original head constraints are enforced by the $\sharp$ -flagged atoms, with $P_1(\sharp, x_2, c, c)$ forcing $x_2 = c$ and $P_1(\sharp, x_3, x_1, x_1)$ forcing x_3 to bind as x_1 , mimicking the repetition of z . Note, however, that these auxiliary atoms live at level 1, whereas the $\sharp$ -facts are stored in D' at level 0: it is the translation of the propagation rules as full copies, transporting $\star$ - and $\sharp$ -atoms alike, that makes them available exactly where the rewritten rule expects them. Indeed, in the minimal model of (D', Σ') the first copy rule provides, at level 1, both the $\star$ -copies of the data and all the $\sharp$ -facts; the intermediate rule then fires with $x_1 \mapsto a, y \mapsto b, x_0 \mapsto \star$ (via $P_1(\sharp, \star, \star, \star)$), $x_2 \mapsto c$ and $x_3 \mapsto a$, deriving $P_2(\star, a, c, a)$. Hence, $\text{ans}(T') = \text{ans}(T) = \{(a)\}$. ■

As illustrated by the previous example, the anonymization operator empties the rule heads of all structural content, constants and repeated variables, and re-encodes it into auxiliary body atoms that consult the $\sharp$ -facts, while the $\star$ flag keeps genuine data apart from this auxiliary machinery; the propagation rules, translated as full copies, transport $\star$ - and $\sharp$ -atoms alike across levels, so that each rewritten rule finds at the level right below its head exactly the $\sharp$ -facts it relies on. Hence an atom $L_i(\bar{a})$ belongs to the minimal model of the input triple if, and only if, $P_i(\star, \bar{a})$ belongs to that of the rewritten one; since the rewritten query is $\star$ -flagged and interrogates the top layer, the answers coincide. The transformation is purely syntactic and increases the size of the triple by at most a polynomial factor.

Proposition 5. *Consider a reasoning triple $T = (q, D, \Sigma)$ in 2TNF. Then, $T \equiv \text{anonymize}(T)$.*

5. Polynomially combined first-order rewriting

The previous sections have normalized the input triple through polynomial-time syntactic transformations preserving query answering. We now present the final rewriting procedure, which takes a 3TNF triple and computes an equivalent first-order rewriting; combined with the normalization steps, it yields the main result of the paper.

Consider a reasoning triple $T = (q(\bar{x}), D, \Sigma)$ in 3TNF obeying the chain $\langle P_0, \dots, P_\ell \rangle$, where χ is the common arity of the predicates and λ is the maximum number of non-frontier variables in a rule of Σ . The query is of the form: $q(\bar{x}) := \exists \bar{y} (P_\ell(\bar{t}_1) \wedge \dots \wedge P_\ell(\bar{t}_n))$, where n is the number of atoms in the query, the sequences $\bar{x}$ and $\bar{y}$ are respectively the free and existential variables of q , and $\bar{t}_i$ (with $1 \leq i \leq n$) is a tuple of terms whose variables are either free or existentially quantified.

Our goal is to design a uniform rewriter $foRew$ that takes as input T and computes, in polynomial time, a first-order pair $(Q'(\bar{x}), D')$ such that $out(Q', D') = ans(q, D, \Sigma)$. More formally:

Proposition 6. *Consider a reasoning triple $T = (q(\bar{x}), D, \Sigma)$ in 3TNF, and let $(Q'(\bar{x}), D') = foRew(T)$. Then, $out(Q', D') = ans(T)$.*

In the rest of this section, we illustrate the construction of $foRew$. We are now ready to combine Propositions 3–6 in order to state the main result of the paper:

Theorem 1. *Consider a reasoning triple $T = (q(\bar{x}), D, \Sigma)$ with $\Sigma \in \text{nrDatalog}$, and let $(Q'(\bar{x}), D') = foRew(\text{anonymize}(\text{layerize}(\text{pad}(T))))$. Then, (Q', D') is computable in polynomial time and $out(Q', D') = ans(T)$.*

5.1. Shaping the first-order pair

We begin by defining the rewritten database D' and the outer interface of the query $Q'(\bar{x})$. The rewritten database is obtained by adding one fact that identifies each rule of the ontology: $D' = D \cup \{\text{RuleId}(c_\sigma) : \sigma \in \Sigma\}$, where, for each $\sigma \in \Sigma$, the fresh constant c_σ serves as a unique identifier for σ . To describe the structure of $Q'(\bar{x})$, we use the following pairwise disjoint sequences of variables: $\bar{x}, \bar{y}, \bar{z}_1, \dots, \bar{z}_\ell, \bar{w}_0, \dots, \bar{w}_\ell$, where $\bar{x}$ and $\bar{y}$ are the free and existential variables of q , while all remaining variables are fresh. Here, n is still the number of atoms in q , and ℓ the index of the last level of the chain $\langle P_0, \dots, P_\ell \rangle$. Moreover, each $\bar{w}_i$ has length χ , while each $\bar{z}_i$ has length λ . The first-order query is then defined as: $Q'(\bar{x}) := \exists \bar{y} (\Phi_\ell(\bar{t}_1) \wedge \dots \wedge \Phi_\ell(\bar{t}_n))$. Here, each $\Phi_\ell(\bar{t}_i)$ is obtained from the single schema formula $\Phi_\ell(\bar{w}_\ell)$ defined in Section 5.3 by renaming its free variables so that $\bar{w}_\ell[j]$ is mapped to $\bar{t}_i[j]$ for every $j \in \{1, \dots, \chi\}$. Consequently, all occurrences of $\Phi_\ell(\bar{t}_i)$ are structurally identical, differing only in the terms replacing the free variables $\bar{w}_\ell$.

5.2. Variable renaming within the ontology

First, we need a variable renaming function ρ that takes as input a rule $\sigma \in \Sigma$ of the form

$$P_{i-1}(\bar{t}_1), \dots, P_{i-1}(\bar{t}_k) \rightarrow P_i(\bar{t}_0)$$

where $\bar{t}_0$ is a sequence of χ distinct variables, and returns the equivalent rule $\rho(\sigma)$ —using only variables taken from the fixed sequences introduced in the previous subsection and constants— of the form:

$$P_{i-1}(\bar{t}_1), \dots, P_{i-1}(\bar{t}_k) \rightarrow P_i(\bar{w}_i)$$

where, for each $j \in \{1, \dots, \chi\}$, we have that each occurrence of $\bar{t}_0[j]$ is replaced by $\bar{w}_i[j]$. Moreover, let $u_1, \dots, u_s$ be the non-frontier variables of σ (with $s \leq \lambda$), each occurrence of u_ℓ is replaced by $\bar{z}_i[\ell]$. Finally, for each $j \in \{1, \dots, \chi\}$, for each $\delta \in \{1, \dots, k\}$, if $\bar{t}_\delta[j]$ is a constant, then $\bar{t}_\delta[j] = \bar{t}_\delta[j]$.

Example 4. Consider the 3TNF triple of Example 3, where $\chi = 4$, $\ell = 2$, and $\lambda = 1$, as the only non-frontier variable of Σ' is the variable y of the intermediate rule. Write $\bar{w}_2 = w_{2,1}, w_{2,2}, w_{2,3}, w_{2,4}$ and $\bar{z}_2 = z_{2,1}$. The renaming ρ maps the intermediate rule to

$$P_1(\star, w_{2,2}, z_{2,1}, c), P_1(\star, c, w_{2,2}, d), P_1(\sharp, w_{2,1}, \star, \star), \\ P_1(\sharp, w_{2,3}, c, c), P_1(\sharp, w_{2,4}, w_{2,2}, w_{2,2}) \rightarrow P_2(\bar{w}_2),$$

and the copy rule of level 2 to $P_1(\bar{w}_2) \rightarrow P_2(\bar{w}_2)$; similarly, the copy rule of level 1 becomes $P_0(\bar{w}_1) \rightarrow P_1(\bar{w}_1)$, and $\bar{z}_1$ remains unused, as copy rules have no non-frontier variables. Each head variable is replaced by the designated variable of $\bar{w}_2$ sitting at its head position, wherever it occurs: x_1 becomes $w_{2,2}$ both in the data atoms and in the auxiliary ones, so the join between x_3 and x_1 resurfaces as the double occurrence of $w_{2,2}$. The non-frontier variable y is sent to $z_{2,1}$, and constants are left untouched. Note the overall effect of the renaming: all rules of level 2 now share literally the same head $P_2(\bar{w}_2)$. This is what allows the formula Φ_2 , constructed next, to treat them as interchangeable alternatives over a shared tuple of variables. ■

5.3. Construction of $\Phi_\ell(\bar{w}_\ell)$

For each rule $\sigma \in \Sigma$ where $\rho(\sigma)$ is of the form (using only variables taken from the fixed sequences introduced in Section 5.1 and constants): $P_{i-1}(\bar{t}_1), \dots, P_{i-1}(\bar{t}_k) \rightarrow P_i(\bar{w}_i)$, first construct the FO-expression $\psi_\sigma \equiv (\bar{w}_{i-1} = \bar{t}_1) \vee \dots \vee (\bar{w}_{i-1} = \bar{t}_k)$. Accordingly, for each $i \in \{1, \dots, \ell\}$, assuming $\sigma_1, \dots, \sigma_\gamma$ are all and only the rules having $P_i(\bar{w}_i)$ in the head, we construct the FO-expression

$$\Phi_i(\bar{w}_i) \equiv \begin{cases} P_0(\bar{w}_0), & \text{if } i = 0, \\ \exists \text{rule}_i \exists \bar{z}_i (\Upsilon_i \wedge \Psi_i(\bar{w}_i)), & \text{if } 1 \leq i \leq \ell. \end{cases}$$

where $\Upsilon_i \equiv (\text{rule}_i = c_{\sigma_1} \vee \dots \vee \text{rule}_i = c_{\sigma_\gamma})$, and where

$$\Psi_i(\bar{w}_i) \equiv \forall \bar{w}_{i-1} [(\psi_{\sigma_1} \wedge \text{rule}_i = c_{\sigma_1}) \vee \dots \vee (\psi_{\sigma_\gamma} \wedge \text{rule}_i = c_{\sigma_\gamma}) \rightarrow \Phi_{i-1}(\bar{w}_{i-1})].$$

In particular, rule_i is a variable that is forced to range over a fixed set of constants which resembles the rules involving the creation of level i .

Note that $\Phi_0, \dots, \Phi_\ell$ are defined by reference: Ψ_i mentions Φ_{i-1} by name, and the actual query is obtained by *unfolding* these references bottom-up, down to the base case $P_0(\bar{w}_0)$. Crucially, Ψ_i contains a *single* occurrence of Φ_{i-1} : the unfolding produces one copy per level rather than branching, so the formula has size bounded by the sum of the sizes of the Φ_i 's, hence polynomial. This is where the folding principle of Section 3 pays off: unfolding the *rules* would instead duplicate Φ_{i-1} once per body atom, reintroducing the blow-up.

Example 5. We continue with the 3TNF triple of Examples 3 and 4. Denote by σ_1 the copy rule of level 1, and by σ_2 and σ_3 the intermediate rule and the copy rule of level 2, respectively. From their renamed versions we obtain

$$\psi_{\sigma_1} \equiv (\bar{w}_0 = \bar{w}_1), \quad \psi_{\sigma_3} \equiv (\bar{w}_1 = \bar{w}_2), \\ \psi_{\sigma_2} \equiv (\bar{w}_1 = (\star, w_{2,2}, z_{2,1}, c)) \vee (\bar{w}_1 = (\star, c, w_{2,2}, d)) \vee (\bar{w}_1 = (\sharp, w_{2,1}, \star, \star)) \\ \vee (\bar{w}_1 = (\sharp, w_{2,3}, c, c)) \vee (\bar{w}_1 = (\sharp, w_{2,4}, w_{2,2}, w_{2,2})).$$

where each tuple equality abbreviates the conjunction of the component-wise ones. Accordingly:

$$\begin{aligned}\Phi_1(\bar{w}_1) &\equiv \exists rule_1 \exists \bar{z}_1 \left(rule_1 = c_{\sigma_1} \wedge \forall \bar{w}_0 [\psi_{\sigma_1} \wedge rule_1 = c_{\sigma_1} \rightarrow P_0(\bar{w}_0)] \right), \\ \Phi_2(\bar{w}_2) &\equiv \exists rule_2 \exists \bar{z}_2 \left((rule_2 = c_{\sigma_2} \vee rule_2 = c_{\sigma_3}) \right. \\ &\quad \left. \wedge \forall \bar{w}_1 [(\psi_{\sigma_2} \wedge rule_2 = c_{\sigma_2}) \vee (\psi_{\sigma_3} \wedge rule_2 = c_{\sigma_3}) \rightarrow \Phi_1(\bar{w}_1)] \right).\end{aligned}$$

The final first-order pair is $Q'(x) = \exists y \Phi_2(\star, x, c, y)$ together with

$$D'' = D' \cup \{\text{RuleId}(c_{\sigma_1}), \text{RuleId}(c_{\sigma_2}), \text{RuleId}(c_{\sigma_3})\}.$$

Let us verify that (a) is returned. Take the witness $y \mapsto a$, so that $\bar{w}_2$ is bound to $(\star, a, c, a)$, and inside Φ_2 guess $rule_2 = c_{\sigma_2}$ and $z_{2,1} \mapsto b$. The antecedent of the guard is then true on exactly five tuples, namely $(\star, a, b, c)$, $(\star, c, a, d)$, $(\sharp, \star, \star, \star)$, $(\sharp, c, c, c)$, $(\sharp, a, a, a)$, which are precisely the body atoms of σ_2 in the derivation of Example 3. On each of them, Φ_1 holds by choosing $rule_1 = c_{\sigma_1}$: its guard requires exactly the corresponding P_0 -fact, and all five belong to D' . Hence $(a) \in \text{out}(Q', D'')$, mirroring $\text{ans}(T') = \{(a)\}$; one can check that no other constant qualifies. ■

6. Conclusion and Future Work

In this paper, we have presented a polynomial-time combined first-order rewriting approach for query answering over non-recursive Datalog knowledge bases. The proposed technique was specifically designed and structured to decompose the rewriting process into distinct sequential stages, thereby proving the theoretical feasibility of the framework and facilitating a clearer understanding of the necessary steps. However, while this pipeline demonstrates that a polynomial-time construction is possible, the current formulation leaves ample room for optimization at multiple points, regarding both the intermediate triple normal forms and the final FO query.

As future work, we aim to systematically optimize the rewriting mechanism to minimize its overall size. Furthermore, we plan to develop an SQL implementation of both this combined technique and the pure UCQ-based rewriting to conduct a thorough comparative experimental evaluation. Finally, we intend to extend this framework to non-recursive existential rules and validate its performance through empirical experiments in this broader setting.

A further open direction concerns determining under which assumptions a polynomial-size pure FO rewriting exists for nrDatalog KBs. In particular, achieving a polynomial bound may depend on: whether quantifiers and equality of the rewritten FO query range strictly over database constants or also include query constants; whether the input CQ and ontology are constant-free or allow explicit constants; and whether the rewritten query can introduce fresh constants or is restricted to the input vocabulary. We plan to systematically investigate the impact of these different settings.

Acknowledgments. The paper has been supported by: **NextGenerationEU** through project FAIR (CUP H23C22000860006); **MIMIT** through the projects ASVIN (CUP B29J24000200005) and PreDiCI (CUP B29J24000300005); **Calabrian Region** through project NextGenGuides (CUP J39I24002040005); **MUR** through project PRODE (CUP H53D23003420006).

Declaration on Generative AI

During the preparation of this work, the authors used Claude and ChatGPT in order to: Paraphrase and reword. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

- [1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
- [2] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.
- [3] M. Arenas, P. Barceló, L. Libkin, W. Martens, A. Pieris, *Database Theory: Querying Data*, Open source manuscript, 2022. Preliminary Version, available at <https://github.com/pdm-book/community>.
- [4] J. Baget, M. Mugnier, S. Rudolph, M. Thomazo, Walking the complexity lines for generalized guarded existential rules, in: *IJCAI, IJCAI/AAAI*, 2011, pp. 712–717.
- [5] M. König, M. Leclère, M. Mugnier, M. Thomazo, Sound, complete and minimal ucq-rewriting for existential rules, *Semantic Web* 6 (2015) 451–475.
- [6] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, A. Pieris, Datalog+/-: A family of logical knowledge representation and query languages for new applications, in: *LICS*, IEEE Computer Society, 2010, pp. 228–242.
- [7] A. Cali, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive relational constraints, *J. Artif. Intell. Res.* 48 (2013) 115–174.
- [8] G. Gottlob, T. Lukasiewicz, A. Pieris, Datalog+/-: Questions and answers, in: *KR, AAAI Press*, 2014.
- [9] N. Leone, M. Manna, G. Terracina, P. Veltri, Efficiently computable datalog $\exists$ programs, in: *KR, AAAI Press*, 2012.
- [10] N. Leone, M. Manna, G. Terracina, P. Veltri, Fast query answering over existential rules, *ACM Trans. Comput. Log.* 20 (2019) 12:1–12:48.
- [11] M. Bienvenu, Ontology-mediated query answering: Harnessing knowledge to get more from data, in: *IJCAI, IJCAI/AAAI Press*, 2016, pp. 4058–4061.
- [12] M. Bienvenu, P. Bourhis, M. Mugnier, S. Tison, F. Ulliana, Ontology-mediated query answering for key-value stores, in: *IJCAI, ijcai.org*, 2017, pp. 844–851.
- [13] P. Barceló, C. Feier, C. Lutz, A. Pieris, When is ontology-mediated querying efficient?, in: *LICS, IEEE*, 2019, pp. 1–13.
- [14] M. M. Dimartino, P. T. Wood, A. Cali, A. Poulouvasilis, Efficient ontology-mediated query answering: Extending dl-liter and linear ELH, *J. Artif. Intell. Res.* 82 (2025) 851–899.
- [15] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, *J. Data Semant.* 10 (2008) 133–173.
- [16] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, D. F. Savo, The MASTRO system for ontology-based data access, *Semantic Web* 2 (2011) 43–53.
- [17] M. Rodriguez-Muro, R. Kontchakov, M. Zakharyashev, Ontology-based data access: Ontop of databases, in: *ISWC (1)*, volume 8218 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 558–573.
- [18] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, M. Zakharyashev, Ontology-based data access: A survey, in: *IJCAI, ijcai.org*, 2018, pp. 5511–5519.
- [19] M. Ortiz, D. Calvanese, T. Eiter, Characterizing data complexity for conjunctive query answering in expressive description logics, in: *AAAI, AAAI Press*, 2006, pp. 275–280.
- [20] T. Eiter, C. Lutz, M. Ortiz, M. Simkus, Query answering in description logics with transitive roles, in: *IJCAI*, 2009, pp. 759–764.
- [21] M. Ortiz, Ontology based query answering: The story so far, in: *AMW*, volume 1087 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2013.
- [22] M. Mugnier, M. Thomazo, An introduction to ontology-based query answering with existential rules, in: *Reasoning Web*, volume 8714 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 245–278.
- [23] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, *J. Autom. Reason.* 39 (2007) 385–429.

- [24] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, *J. Web Semant.* 14 (2012) 57–83.
- [25] G. Gottlob, G. Orsi, A. Pieris, Query rewriting and optimization for ontological databases, *ACM Trans. Database Syst.* 39 (2014) 25:1–25:46.
- [26] G. Gottlob, S. Kikot, R. Kontchakov, V. V. Podolskii, T. Schwentick, M. Zakharyashev, The price of query rewriting in ontology-based data access, *Artif. Intell.* 213 (2014) 42–59.
- [27] M. Bienvenu, S. Kikot, R. Kontchakov, V. V. Podolskii, M. Zakharyashev, Ontology-mediated queries: Combined complexity and succinctness of rewritings via circuit complexity, *J. ACM* 65 (2018) 28:1–28:51.
- [28] B. Rossman, Homomorphism preservation theorems, *J. ACM* 55 (2008) 15:1–15:53.
- [29] C. Lutz, D. Toman, F. Wolter, Conjunctive query answering in the description logic EL using a relational database system, in: *IJCAI*, 2009, pp. 2070–2075.
- [30] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyashev, The combined approach to query answering in dl-lite, in: *KR*, AAAI Press, 2010.
- [31] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyashev, The combined approach to ontology-based data access, in: *IJCAI*, *IJCAI/AAAI*, 2011, pp. 2656–2661.
- [32] G. Gottlob, M. Manna, A. Pieris, Polynomial combined rewritings for existential rules, in: *KR*, AAAI Press, 2014.
- [33] G. Gottlob, M. Manna, A. Pieris, Polynomial rewritings for linear existential rules, in: *IJCAI*, AAAI Press, 2015, pp. 2992–2998.
- [34] G. Gottlob, M. Manna, A. Pieris, Polynomial combined rewritings for linear existential rules and dl-lite with n-ary relations, in: *Description Logics*, volume 1350 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015.
- [35] G. Gottlob, M. Manna, A. Pieris, Polynomial combined first-order rewritings for linear and guarded existential rules, *Artif. Intell.* 321 (2023) 103936.
- [36] G. Gottlob, T. Schwentick, Rewriting ontological queries into small nonrecursive datalog programs, in: *KR*, AAAI Press, 2012.
- [37] S. Kikot, R. Kontchakov, V. V. Podolskii, M. Zakharyashev, Exponential lower bounds and separation for query rewriting, in: *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 263–274.
- [38] P. Pudlák, The lengths of proofs, in: S. R. Buss (Ed.), *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, North-Holland, Amsterdam, 1998, pp. 547–637.
- [39] J. Avigad, Eliminating definitions and skolem functions in first-order logic, in: *16th Annual IEEE Symposium on Logic in Computer Science (LICS 2001)*, IEEE Computer Society, 2001, pp. 139–146.
- [40] M. J. Fischer, M. O. Rabin, Super-exponential complexity of Presburger arithmetic, in: *Complexity of Computation*, volume 7 of *SIAM–AMS Proceedings*, 1974, pp. 27–41.
- [41] M. Sipser, *Introduction to the Theory of Computation*, 3 ed., Cengage Learning, 2012.