Sufficient Conditions for First-Order and Datalog Rewritability in $\mathcal{ELU}$

Mark Kaminski and Bernardo Cuenca Grau

Department of Computer Science
University of Oxford, UK.

Abstract. We study the problem of answering instance queries over non-Horn ontologies by rewriting them into Datalog programs or First-Order queries (FOQs). We consider the basic non-Horn language $\mathcal{ELU}$, which extends $\mathcal{EL}$ with disjunctions. Both Datalog-rewritability and FO-rewritability of instance queries have been recently shown to be decidable for $\mathcal{ALC}$ ontologies (and hence also for $\mathcal{ELU}$); however, existing decision methods are mainly of theoretical interest. We identify two fragments of $\mathcal{ELU}$ for which we can compute Datalog-rewritings and FO-rewritings of instance queries, respectively, by means of a resolution-based algorithm.

1 Introduction

We study the problem of answering queries over DL ontologies by rewriting them into a First-Order query (FOQ) or a Datalog program. On the theoretical side, rewriting queries into an FOQ or a Datalog program ensures tractability of query answering in terms of data complexity. On the practical side, it allows one to reuse optimised data management systems, namely RDBMSs in the case of FOQs and rule-based systems such as OWLim or Oracle's Semantic Data Store in the case of Datalog.

The problem of ontology-based query answering via query rewriting has so far been mainly studied for Horn DLs. FO-rewritability is ensured for logics of the DL-Lite family [5] and query rewriting algorithms in these DLs have been implemented in systems such as QuOnto [1], Presto [16], Quest [15], Rapid [6], and Owlgres [19]. Datalog rewritability is ensured for logics of the $\mathcal{EL}$ family, as well as more expressive languages such as Horn-$\mathcal{SHIQ}$ [11]; optimised algorithms have been implemented in systems such as Requiem [14] and Clipper [10]. FO-rewritability has also been studied as a decision problem for logics of the $\mathcal{EL}$-family [3], where tight complexity bounds have been shown.

Horn DLs, however, cannot capture disjunctive knowledge, such as 'every student is either a graduate or an undergraduate'. As a consequence, ontologies capturing disjunctive knowledge cannot be processed using existing rewriting techniques. Little is known about how to compute FO and Datalog rewritings for non-Horn ontologies, and first results have been obtained only recently. Instance queries (i.e., queries of the form $A(x)$ with A a concept name) are known to be FO-rewritable w.r.t. ontologies expressed in certain logics of the DL-Lite$_{\mathsf{bool}}$ family —the extension of DL-Lite logics with disjunction [2]—, and a goal-oriented

resolution algorithm for computing rewritings w.r.t. such logics has been proposed in [8]. In contrast, instance queries w.r.t. ontologies expressed in the basic non-Horn DL $\mathcal{ELU}$ are not generally rewritable into Datalog: a result that holds regardless of any complexity-theoretic assumptions [8]. While FO and Datalog rewritability of instance queries w.r.t. $\mathcal{ALC}$ ontologies have been proved to be decidable [4], the decision methods in [4] are problematic in practice as their first step, translation to CSP, has exponential best-case complexity. Finally, for certain classes of conjunctive queries, query answering over non-Horn ontologies can be reduced to knowledge base consistency [**?**]. This approach, however, does not generally ensure tractability in terms of data complexity.

In this paper, we are interested in ontologies formulated in the basic non-Horn DL $\mathcal{ELU}$. For simplicity, we focus on instance queries of the form $A(x)$. In general, however, our results apply to *ground queries*, where all variables are required to be mapped to constants; such queries form the basis of the standard query language SPARQL. Our main result is the identification of two sufficient conditions on $\mathcal{ELU}$-ontologies that ensure FO-rewritability and Datalog rewritability of instance queries, respectively. Furthermore, we provide resolution-based algorithms that can be used for computing the corresponding rewritings in case our sufficient conditions are fulfilled. Our algorithms build on the generic resolution-based technique proposed in [8].

This paper is accompanied with an appendix containing the proofs of our technical results.

2 Preliminaries

We consider First-Order logic without equality and function symbols. Variables, terms, (ground) atoms, literals, formulae, sentences, intepretations, models and entailment are defined as usual. An *ABox* is a finite set of ground atoms (called facts). We also adopt standard notions of (Horn) clauses, (variable) substitutions, and most general unifiers (MGUs). *Positive factoring* (PF) and *binary resolution* (BR) are as follows, where σ is the MGU of atoms A and B:

$$\text{PF:}\quad \frac{C \vee A \vee B}{C\sigma \vee A\sigma} \qquad \text{BR:}\quad \frac{C \vee A \quad D \vee \neg B}{(C \vee D)\sigma}$$

A clause C is a *tautology* if it contains literals A and $\neg A$. A clause C subsumes a clause D if a substitution σ exists such that each literal in $C\sigma$ occurs in D. Furthermore, C θ-subsumes D if C subsumes D and C has no more literals than D. Clause C is *redundant* in a set of clauses if C is a tautology or if C is θ-subsumed by another clause in the set. The *condensation* of a clause C is the clause D with the least number of literals such that $D \subseteq C$ and C subsumes D.

The Description Logic $\mathcal{ELU}$. We assume familiarity with standard DLs; here, we briefly recapitulate the syntax of $\mathcal{ELU}$. An $\mathcal{ELU}$-concept is an expression of the form $\top$, A, $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, or $\exists R.C$, where A is a concept name, $C_{(i)}$ are concepts, and R is a role name. An $\mathcal{ELU}$-TBox $\mathcal{T}$ is a finite set of GCIs of the form $C_1 \sqsubseteq C_2$ where C_1 and C_2 are $\mathcal{ELU}$-concepts. An $\mathcal{ELU}$-TBox is

normalised it it contains only axioms of the form $\exists R.A \sqsubseteq B$, $A \sqsubseteq \exists R.B$, and $\bigsqcap_{i=1}^{n} A_i \sqsubseteq \bigsqcup_{j=1}^{m} B_j$, where $A_{(i)}$ and $B_{(j)}$ are either named or $\top$. Each $\mathcal{ELU}$-TBox $\mathcal{T}$ can be transformed in polynomial time into a normalised $\mathcal{ELU}$-TBox that is a conservative extension of $\mathcal{T}$. An $\mathcal{ELU}$-TBox $\mathcal{T}$ is *linear* if conjunction $\sqcap$ does not occur on the left-hand side of a GCI in $\mathcal{T}$.

Rules. A rule is a First-Order sentence of the form $\forall \boldsymbol{x}.\forall \boldsymbol{z}.[\varphi(\boldsymbol{x}, \boldsymbol{z}) \rightarrow \psi(\boldsymbol{x})]$, where tuples of variables $\boldsymbol{x}$ and $\boldsymbol{z}$ are disjoint, $\varphi(\boldsymbol{x}, \boldsymbol{z})$ is a conjunction of atoms, and $\psi(\boldsymbol{x})$ is a disjunction of atoms. Formula φ is the *body* of r, formula ψ is the *head* of r, and quantifiers in a rule are omitted for brevity. Furthermore, we often abuse notation and treat a rule and its equivalent clause as synonyms. A rule is *Datalog* if $\psi(\boldsymbol{x})$ is a single atom, and it is *disjunctive* otherwise. A (Datalog) program is a finite set of (Datalog) rules. A program is *monadic* if every predicate occurring in the head of a rule is unary.

An $\mathcal{ELU}$-program consists of the following kinds of rules: *(i)* $\bigwedge_{i=1}^{n} A_i(x) \rightarrow \bigvee_{j=1}^{m} B_j(x)$ and *(ii)* $R(x, y) \wedge A(y) \rightarrow B(x)$, where the atom $A(y)$ can be omitted. For convenience, we use the unary atom $\top(x)$ to denote an empty rule body. An $\mathcal{EL}$-program is an $\mathcal{ELU}$-program that is also Datalog. Finally, an $\mathcal{ELU}$-program is linear if conjunction does not occur on the left-hand side of a rule of type *(i)*.

Queries. An (instance) query is a unary atom $Q(x)$. A constant c is an answer to $Q(x)$ w.r.t. a set of FO sentences $\mathcal{F}$ and an ABox $\mathcal{A}$ if $\mathcal{F} \cup \mathcal{A} \models Q(c)$. The set of answers to Q relative to $\mathcal{F}$ and $\mathcal{A}$ is denoted as $\mathsf{cert}(Q(x), \mathcal{F}, \mathcal{A})$.

FO and Datalog Rewritings. For an ABox $\mathcal{A}$, let $I_{\mathcal{A}}$ the interpretation corresponding to $\mathcal{A}$ in the obvious way. An FO formula $\varphi(x)$ with one free variable is an FO-rewriting of a query $Q(x)$ w.r.t. an $\mathcal{ELU}$-TBox (or, equivalently, an $\mathcal{ELU}$-program) $\mathcal{T}$ if the following condition holds for every constant c and ABox $\mathcal{A}$: $c \in \mathsf{cert}(Q, \mathcal{T}, \mathcal{A})$ iff $I_{\mathcal{A}} \models \varphi(c)$. Furthermore, we say that $\varphi(x)$ is a UCQ-rewriting if it is of the form $\varphi(x) = \bigvee_{i=1}^{n} \varphi_i(x)$ where each $\varphi_i(x)$ is constructed using only conjuction and existential quantification. We say that $Q(x)$ is FO-rewritable w.r.t. $\mathcal{T}$ if an FO-rewriting of $Q(x)$ w.r.t. $\mathcal{T}$ exists. Finally, $\mathcal{T}$ is FO-rewritable if for each concept name A in $\mathcal{T}$ the query $A(x)$ is FO-rewritable w.r.t. $\mathcal{T}$. A Datalog program $\mathcal{P}$ is a rewriting of a query $Q(x)$ relative to a TBox (or, equivalently, an $\mathcal{ELU}$-program) $\mathcal{T}$ if $\mathsf{cert}(Q(x), \mathcal{T}, \mathcal{A}) = \mathsf{cert}(Q(x), \mathcal{P}, \mathcal{A})$ for every ABox $\mathcal{A}$. We say that $Q(x)$ is Datalog-rewritable w.r.t. $\mathcal{T}$ if a Datalog rewriting of $Q(x)$ w.r.t. $\mathcal{T}$ exists. Finally, $\mathcal{P}$ is a rewriting of $\mathcal{T}$ if it is a rewriting for each query $A(x)$ w.r.t. $\mathcal{T}$, with A a concept name in $\mathcal{T}$; TBox $\mathcal{T}$ is Datalog-rewritable if a Datalog rewriting of $\mathcal{T}$ exists. As observed in [4], it follows from the homomorphism preservation theorem for finite structures [17] that each FO-rewritable query is also UCQ rewritable and hence Datalog rewritable as well.

3 Computing Rewritings via Resolution

We next recapitulate the generic resolution-based technique proposed in [8], which takes a TBox $\mathcal{T}$ and attempts to rewrite it into a Datalog program. If $\mathcal{T}$ is restricted to be in $\mathcal{ELU}$, this technique consists of the following two steps.

Procedure 1 Compile-Horn

Input: $\mathcal{S}$: set of clauses
Output: $\mathcal{S}_H$: set of Horn clauses

$\mathcal{S}_H := \{C \in \mathcal{S} \mid C$ is a Horn clause and not a tautology$\}$
$\mathcal{S}_{\overline{H}} := \{C \in \mathcal{S} \mid C$ is a non-Horn clause and not a tautology$\}$
repeat
  $\mathcal{F}$:= factors of each $C_1 \in \mathcal{S}_{\overline{H}}$ non-redundant in $\mathcal{S}_H \cup \mathcal{S}_{\overline{H}}$
  $\mathcal{R}$: = resolvents of each $C_1 \in \mathcal{S}_{\overline{H}}$ and $C_2 \in \mathcal{S}_{\overline{H}} \cup \mathcal{S}_H$ not redundant in $\mathcal{S}_H \cup \mathcal{S}_{\overline{H}}$
  for each $C \in \mathcal{F} \cup \mathcal{R}$ **do**
    C' := the condensation of C
    Delete from $\mathcal{S}_H$ and $\mathcal{S}_{\overline{H}}$ all clauses θ-subsumed by C'
    if C' is Horn **then** $\mathcal{S}_H := \mathcal{S}_H \cup \{C'\}$
    else $\mathcal{S}_{\overline{H}} := \mathcal{S}_{\overline{H}} \cup \{C'\}$
until $\mathcal{F} \cup \mathcal{R} = \emptyset$
return $\mathcal{S}_H$

Step 1: From DLs to Disjunctive Datalog. First, the algorithm in [12] is applied to transform $\mathcal{T}$ into an $\mathcal{ELU}$-program $\mathcal{D}$ that entails the same facts as $\mathcal{T}$ w.r.t. every ABox. By eliminating the positive occurrences of existential quantifiers, one thus reduces the problem of rewriting the DL TBox $\mathcal{T}$ to the problem of rewriting the disjunctive program $\mathcal{D}$.

Step 2: From Disjunctive Datalog to Datalog. Second, the program $\mathcal{D}$ is transformed into a Datalog program $\mathcal{P}$ using a variant of the mainstream knowledge compilation algorithms proposed in [18] for propositional clauses, and extended in [9] to First-Order clauses (but without termination guarantees). Procedure 1 summarises (a slight variation of) the technique from [9]. Roughly speaking, the procedure applies binary resolution and factoring and keeps only the consequences that are not redundant. Unlike unrestricted resolution, the procedure never resolves two Horn clauses. The main property of Procedure 1 shown in [9] is that, even if it never terminates, each Horn consequence of the input $\mathcal{S}$ will also eventually become entailed by $\mathcal{S}_H$. In [8], it was shown that Procedure 1 also enjoys the following key property: if the program $\mathcal{D}$ is given as input and the procedure terminates, then the output $\mathcal{P}$ is a Datalog rewriting of $\mathcal{D}$. In essence, this result implies that compilation into Horn clauses can be done in an ABox independent way: $\mathcal{D}$ and $\mathcal{P}$ can be combined with an arbitrary ABox and still entail the same facts.

Example 3.1 *We use the $\mathcal{ELU}$-programs $\mathcal{D}^1$ and $\mathcal{D}^2$ as running examples:*

$$\mathcal{D}^1 = \{\ C(x) \to A(x) \vee B(x) \qquad R(x,y) \wedge A(y) \to H(x) \qquad R(x,y) \wedge B(y) \to D(x) \qquad R(x,y) \wedge D(y) \to H(x)\ \}$$

$$\mathcal{D}^2 = \{\ \top(x) \to A(x) \vee B(x) \qquad R(x,y) \wedge A(y) \to B(x) \qquad R(x,y) \wedge B(y) \to A(x)\ \}$$

When applied to program $\mathcal{D}^1$, Procedure 1 terminates with the following sets $\mathcal{D}^1_{\overline{H}}$ and $\mathcal{D}^1_H$ of disjunctive and Datalog rules, respectively:

$$\mathcal{D}^1_{\overline{H}} = \{ \quad \underline{C(x) \to A(x) \vee B(x)} \qquad \mathcal{D}^1_H = \{ \; \underline{R(x,y) \wedge A(y) \to H(x)}$$
$$R(x,y) \wedge C(y) \to H(x) \vee B(y) \qquad \underline{R(x,y) \wedge B(y) \to D(x)}$$
$$R(x,y) \wedge C(y) \to D(x) \vee A(y) \qquad \underline{R(x,y) \wedge D(y) \to H(x)}$$
$$R(x,y) \wedge R(z,y) \wedge C(y) \to H(x) \vee D(z) \; \} \qquad R(x,z) \wedge R(x,y) \wedge R(z,y) \wedge C(y) \to H(x) \; \}$$

The results proved in [8] imply that $\mathcal{D}^1_H$ is a Datalog rewriting of $\mathcal{D}^1$. In contrast, Procedure 1 does not terminate when given $\mathcal{D}^2$ as input. Although the mutually recursive Datalog rules in $\mathcal{D}^2$ are never resolved directly with each other, they can interact via the program's (only) disjunctive rule. As a result, Procedure 1 will eventually derive each of the following (infinitely many) rules, for each even number n and for each $Z \in \{A, B\}$:

$$R(x_n, x_0) \wedge \bigwedge_{i=1}^{n} R(x_i, x_{i-1}) \to Z(x_n)$$

4 Sufficient Conditions for Rewritability

In what follows, we present two conditions on $\mathcal{ELU}$-programs that ensure FO and Datalog-rewritability, respectively. Both conditions come with resolution-based algorithms for computing rewritings. To ensure termination of resolution, both our conditions apply to linear $\mathcal{ELU}$-programs only. As we discuss later on in §5, termination in the non-linear case becomes much more problematic, and it is left for future work.

4.1 FO Rewritability

As shown by Bienvenu et al. [3], if an instance query $A(x)$ is FO-rewritable w.r.t. an $\mathcal{EL}$-TBox, then there exists a tree-shaped UCQ that is an FO-rewriting for the given query and TBox. This property was critical to the study of FO-rewritability in the context of $\mathcal{EL}$ as it implies a characterisation of FO-rewritability in terms of tree-shaped ABoxes. Since only tree-shaped ABoxes are relevant, one can exploit standard tree automata machinery to study the problem.

Once disjunctions come into play, however, this key property seems to break. As demonstrated by the following example, there are FO-rewritable $\mathcal{ELU}$-TBoxes that do not seem to be rewritable into tree-shaped UCQs. Thus, the machinery developed in [3] for $\mathcal{EL}$ does not seem to extend to $\mathcal{ELU}$.

Example 4.1 *The query $H(x)$ is FO-rewritable w.r.t. our example program $\mathcal{D}^1$. The following UCQ $\varphi(x) = \varphi_1(x) \vee \varphi_2(x) \vee \varphi_3(x) \vee \varphi_4(x) \vee \varphi_5(x)$ can be obtained*

by unfolding the Datalog rewriting obtained by Procedure 1 on $\mathcal{D}^1$*:*

$$\varphi_1(x) = H(x) \qquad \varphi_4(x) = \exists y.\exists z.\, R(x,y) \wedge R(x,z) \wedge R(z,y) \wedge C(y)$$
$$\varphi_2(x) = \exists y.\, R(x,y) \wedge A(y) \qquad \varphi_5(x) = \exists y.\exists z.\, R(x,y) \wedge R(y,z) \wedge B(z)$$
$$\varphi_3(x) = \exists y.\, R(x,y) \wedge D(y)$$

Clearly, $\varphi_4(x)$ *is not tree-shaped. Moreover,* $H(x)$ *does not seem to have a tree-shaped UCQ-rewriting as defined in [3].*

Although a linear $\mathcal{ELU}$-program $\mathcal{D}$ that is FO-rewritable may not have a tree-shaped rewriting, linearity allows us to restrict resolution proofs in a critical way. We can show that each derivation from $\mathcal{D}$ via resolution and condensation only (i.e., no factoring) involves only tree-shaped rules. This implies that each non-tree rule derived by resolution and factoring is a factor of a tree-shaped rule.

In the following, we define a condition on linear $\mathcal{ELU}$ programs that guarantees FO-rewritability. When applied to a program $\mathcal{D}$ satisfying the condition (such as $\mathcal{D}^1$ in our running example), Procedure 1 will terminate: the size of derived tree-shaped rules will be limited by the size of $\mathcal{D}$. Furthermore, the Datalog program obtained as output will be *bounded* for every predicate in the standard sense [13]. Our condition is defined as given next.

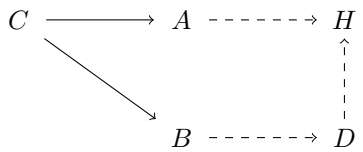
Definition 4.2. *The* dependency graph *of a linear* $\mathcal{ELU}$*-program* $\mathcal{D}$ *is the least directed edge-labeled graph* $G_{\mathcal{D}} = (V, E, \mu)$ *satisfying the following conditions:*

1. *each unary predicate occurring in* $\mathcal{D}$ *is a node in* V*;*
2. $(A, B) \in E \cap \mu$ *for each rule in* $\mathcal{D}$ *of the form* $R(x,y) \wedge A(y) \rightarrow B(x)$*;*
3. $(A, B_i) \in E$ *with* $i \in [1,n]$ *for each rule in* $\mathcal{D}$ *of the form* $A(x) \rightarrow \bigvee_{i=1}^{n} B_i(x)$*.*

Edges contained in the labelling μ *are called* transfer edges*; all remaining edges are called* propositional edges*. A* transfer cycle *is a simple cycle that contains at least one transfer edge. The program* $\mathcal{D}$ *is acyclic if* $G_{\mathcal{D}}$ *contains no transfer cycle. Finally, the* transfer depth *of a unary predicate* A *in an acyclic program* $\mathcal{D}$ *is the maximal number of transfer edges on a path in* $G_{\mathcal{D}}$ *ending in* A*.*

Intuitively, the maximal transfer depth of a predicate in an acyclic program establishes a bound on the role depth of the tree-shaped rules that can be generated by Procedure 1. Termination of Procedure 1 is then ensured by condensation, which bounds the branching factor of rules in terms of their depth.

Example 4.3 *Consider the program* $\mathcal{D}^1$ *from Example 3.1. The dependency graph for* $\mathcal{D}^1$ *is given next, where transfer edges are dashed. Clearly,* $\mathcal{D}^1$ *is acyclic and predicate* H *has transfer depth* 2*.*



Procedure 2 Rewrite-UCQ

Input: $\mathcal{D}$: a linear, acyclic $\mathcal{ELU}$-program;
A: a unary predicate occurring in $\mathcal{D}$
Output: $\varphi(x)$: a UCQ

1: $\mathcal{P} := \mathsf{Compile\text{-}Horn}(\mathcal{D})$
2: $\mathcal{P}' := \{A(x) \to Q(x)\}$ where Q is a fresh unary predicate
3: $\mathcal{P}'' := \emptyset$
4: **repeat**
5: Select some $r \in \mathcal{P}'$
6: $\mathcal{N} :=$ the resolvents of r with clauses in $\mathcal{P}$ that are not subsumed in $\mathcal{P}' \cup \mathcal{P}''$
7: $\mathcal{P}' := (\mathcal{P}' \setminus \{r\}) \cup \mathcal{N}$
8: $\mathcal{P}'' := \mathcal{P}'' \cup \{r\}$
9: **until** $\mathcal{P}' = \emptyset$
10: $\varphi(x) := \bigvee\{\, \exists \boldsymbol{y}.\, \psi(x, \boldsymbol{y}) \mid \forall x.\, \forall \boldsymbol{y}.\, \psi(x, \boldsymbol{y}) \to Q(x) \in \mathcal{P}'' \,\}$
11: **return** $\varphi(x)$

Rules derived by Procedure 1 on $\mathcal{D}^1$ have depth at most 2 (see Example 3.1).

Procedure 2 computes a UCQ rewriting of a query $A(x)$ relative to an acyclic program $\mathcal{D}$. On input $\mathcal{D}$ and A, we first apply Procedure 1 to compute a Datalog rewriting $\mathcal{P}$ of $\mathcal{D}$; as already seen, termination of this first step is guaranteed. Predicate A is then unfolded in $\mathcal{P}$ using standard techniques, as shown in Lines (2-10); unfolding terminates iff the predicate A is bounded in $\mathcal{P}$ [13, 7]. To show termination of unfolding, we observe that the depth of the rules in $\mathcal{P}$ still cannot exceed the transfer depth of the predicates in the dependency graph of $\mathcal{D}$; hence, the loop in Lines (4)-(9) only derives finitely many rules modulo subsumption. The properties of the procedure are thus as follows.

Theorem 4.4. *Given an acyclic $\mathcal{ELU}$-program $\mathcal{D}$ and a unary predicate A in $\mathcal{D}$, Procedure 2 terminates and returns a UCQ rewriting of $A(x)$ w.r.t. $\mathcal{D}$.*

Example 4.5 *When applied to the predicate H and the program $\mathcal{D}^1$ from Example 3.1, Procedure 2 first calls Procedure 1, which returns the program $\mathcal{D}^1_H$ from Example 3.1 as a Datalog rewriting of $\mathcal{D}^1$. So, $\mathcal{P}$ is initialised with $\mathcal{D}^1_H$ and $\mathcal{P}'$ is set to $\{H(x) \to Q(x)\}$. On these values, unfolding terminates after five iterations of the main loop with the following rules in $\mathcal{P}''$:*

$$H(x) \to Q(x) \qquad R(x,z) \wedge R(x,y) \wedge R(z,y) \wedge C(y) \to Q(x)$$
$$R(x,y) \wedge A(y) \to Q(x) \qquad R(x,y) \wedge R(y,z) \wedge B(z) \to Q(x)$$
$$R(x,y) \wedge D(y) \to Q(x)$$

This yields precisely the UCQ $\varphi(x)$ given in Example 4.1.

4.2 Datalog Rewritability

Our Datalog-rewritability condition relaxes the acyclicity condition from §4.1 by allowing certain kinds of cyclic programs. Intuitively, we allow programs that

can be separated into two parts: one that may contain disjunctive rules but no transfer cycles, and another one that contains only Datalog rules, but may contain transfer cycles. We call such programs *separable*.

Definition 4.6. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program and let $\mathcal{D}_\vee$ be the smallest subset of $\mathcal{D}$ such that*

1. *each disjunctive rule in $\mathcal{D}$ is contained in $\mathcal{D}_\vee$; and*
2. *no unary predicate in $\mathcal{D} \setminus \mathcal{D}_\vee$ occurs in the body of a rule in $\mathcal{D}_\vee$.*

Program $\mathcal{D}$ is separable *if $\mathcal{D}_\vee$ is acyclic.*

Example 4.7 *Program $\mathcal{D}^2$ from our running example 3.1 is not FO-rewritable; however, it is separable: $\mathcal{D}^2_\vee = \{\top(x) \to A(x) \vee B(x)\}$ and it is therefore acyclic as in Definition 4.2.*

We deal with separable programs by eliminating cycles from their Datalog part. Cycle elimination is based on the observation that every linear $\mathcal{EL}$-program $\mathcal{P}$ can be seen as a labeled transition system $T_\mathcal{P}$ with unary predicates as states and rules as transitions labeled by binary predicates (or $\top$ if the rule contains no binary predicate). Given a unary predicate B in $\mathcal{P}$, an ABox $\mathcal{A}$, and an individual b, every derivation of $B(b)$ from $\mathcal{P} \cup \mathcal{A}$ corresponds to a path ending in B in $T_\mathcal{P}$; hence, paths in $\mathcal{T}_P$ encode resolution proofs.

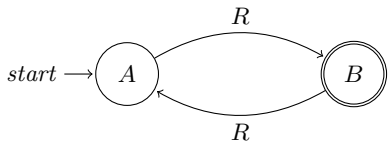
The set of all possible derivations of an assertion $B(b)$ from an assertion $A(a)$ by the rules in $\mathcal{P}$ corresponds to the regular language accepted by the non-deterministic finite automaton constructed from $\mathcal{T}_\mathcal{P}$ by taking A as the unique initial state, and B as the unique accepting state.

Definition 4.8. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program and let $\langle A, B\rangle$ be a pair of unary predicates occurring in $\mathcal{D}$. The automaton for $\mathcal{D}$ relative to $\langle A, B\rangle$ is the non-deterministic finite word automaton $\mathsf{Aut}_\mathcal{D}(A, B) = \langle \mathbf{S}, \Gamma, \to, \{A\}, \{B\}\rangle$ defined as follows: the set of states $\mathbf{S}$ consists of the unary predicates in $\mathcal{D}$; the alphabet Γ is the set of binary predicates in $\mathcal{D}$ plus the special symbol $\top$; the (unique) initial and final states are A and B respectively; finally, the transition relation $\to$ consists of the following transitions:*

- *$C \to_\top D$ for each $\mathcal{EL}$-rule $C(x) \to D(x)$ in $\mathcal{D}$;*
- *$C \to_R D$ for each $\mathcal{EL}$-rule $R(x, y) \wedge C(y) \to D(x)$ in $\mathcal{D}$.*

Finally, we denote with $\mathcal{L}(\mathsf{Aut}_\mathcal{D}(A, B))$ the language accepted by $\mathsf{Aut}_\mathcal{D}(A, B)$.

Example 4.9 *The automaton for $\mathcal{D}^2$ relative to $\langle A, B\rangle$ is given below:*



The automata for $\langle A, A\rangle$, $\langle B, A\rangle$, and $\langle B, B\rangle$ contain exactly the same states and transitions and only differ on the particular choice of initial and final state.

The language $\mathcal{L}(\mathsf{Aut}_{\mathcal{D}}(A, B))$ is regular, and hence can be described by means of some regular expression over the alphabet Γ of the automaton. We adopt the following definition of regular expressions over Γ, where $\alpha \in \Gamma$ and the atomic expression $\emptyset$ denotes the empty language.

$$e \ ::= \ \alpha \mid \emptyset \mid ee \mid e + e \mid e^*$$

With each regular expression e over Γ we uniquely associate a (fresh) binary predicate N_e and a Datalog program $\mathcal{P}_e$ that defines N_e as shown next.

Definition 4.10. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program and let e be a regular expression over the binary predicates in $\mathcal{D}$ and the symbol $\top$. The Datalog program $\mathcal{P}_e$ corresponding to e is defined inductively as follows:*

$$\begin{aligned}
\mathcal{P}_\emptyset &= \emptyset \\
\mathcal{P}_\top &= \{\top(x) \to N_\top(x, x)\} \\
\mathcal{P}_R &= \{R(x, y) \to N_R(y, x)\} \\
\mathcal{P}_{e_1 e_2} &= \mathcal{P}_{e_1} \cup \mathcal{P}_{e_2} \cup \{N_{e_1}(x, y) \wedge N_{e_2}(y, z) \to N_{e_1 e_2}(x, z)\} \\
\mathcal{P}_{e_1 + e_2} &= \mathcal{P}_{e_1} \cup \mathcal{P}_{e_2} \cup \{N_{e_1}(x, y) \to N_{e_1+e_2}(x, y), N_{e_2}(x, y) \to N_{e_1+e_2}(x, y)\} \\
\mathcal{P}_{e^*} &= \mathcal{P}_e \cup \{\top(x) \to N_{e^*}(x, x),\ N_e(x, y) \wedge N_{e^*}(y, z) \to N_{e^*}(x, z)\}
\end{aligned}$$

Example 4.11 *The language of the automaton $\mathsf{Aut}_{\mathcal{D}^2}(A, B)$ in Example 4.9 can be captured by the regular expression $e_{\langle A,B\rangle} = R(RR)^*$. The corresponding program $\mathcal{P}_{e_{\langle A,B\rangle}}$ consists of the following Datalog rules:*

$$R(x, y) \to N_R(y, x) \tag{1}$$
$$N_R(x, y) \wedge N_R(y, z) \to N_{RR}(x, z) \tag{2}$$
$$\top(x) \to N_{(RR)^*}(x, x) \tag{3}$$
$$N_{RR}(x, y) \wedge N_{(RR)^*}(y, z) \to N_{(RR)^*}(x, z) \tag{4}$$
$$N_R(x, y) \wedge N_{(RR)^*}(y, z) \to N_{R(RR)^*}(x, z) \tag{5}$$

The language for $\mathsf{Aut}_{\mathcal{D}^2}(A, A)$ is captured by $e_{\langle A,A\rangle} = (RR)^*$, which is a subexpression of $e_{\langle A,B\rangle}$; the program $\mathcal{P}_{e_{\langle A,A\rangle}}$ thus consists of rules (1)-(4). Symmetry in the automaton's transitions implies $\mathcal{P}_{e_{\langle B,A\rangle}} = \mathcal{P}_{e_{\langle A,B\rangle}}$ and $\mathcal{P}_{e_{\langle B,B\rangle}} = \mathcal{P}_{e_{\langle A,A\rangle}}$.

Let $\mathcal{D}$ be a separable program and $\mathcal{D}_\vee$ be the disjunctive part of $\mathcal{D}$. Since for each pair A, B in $\mathcal{D} \setminus \mathcal{D}_\vee$ we have that $\mathcal{L}(\mathsf{Aut}_{\mathcal{D}}(A, B))$ can be described by some regular expression $e_{\langle A,B\rangle}$, all ways in which an assertion $A(a)$ can imply $B(b)$ can be summarised by the single rule $N_{e_{\langle A,B\rangle}}(x, y) \wedge A(x) \to B(y)$. Thus, the disjunctive program $\mathcal{D}'$ defined as the union of $\mathcal{D}_\vee$, all programs $\mathcal{P}_{\langle A,B\rangle}$, and all rules $N_{e_{\langle A,B\rangle}}(x, y) \wedge A(x) \to B(y)$ has the same Horn consequences as $\mathcal{D}$. To obtain such Horn consequences, we could thus apply Procedure 1 to $\mathcal{D}'$; however, to ensure termination we do the following instead:

Procedure 3 Rewrite-Datalog

Input: $\mathcal{D}$: a separable $\mathcal{ELU}$-program;
Output: $\mathcal{P}$: a Datalog program

1: $\Xi(\mathcal{D}), \Omega(\mathcal{D}) := \emptyset$
2: **for each** pair of unary predicates $\langle A, B\rangle$ **do**
3: $\quad \mathsf{Aut}_{\mathcal{D}}(A, B) :=$ finite word automaton for $\mathcal{D}$ relative to $\langle A, B\rangle$
4: $\quad$ Construct a regular expression $e_{\langle A,B\rangle}$ equivalent to $\mathsf{Aut}_{\mathcal{D}}(A, B)$
5: $\quad$ Construct Datalog program $\mathcal{P}_{e_{\langle A,B\rangle}}$ and
6: $\quad \Xi(\mathcal{D}) := \Xi(\mathcal{D}) \cup \mathcal{P}_{e_{\langle A,B\rangle}}$
7: $\quad \Omega(\mathcal{D}) := \Omega(\mathcal{D}) \cup \{R_{e_{\langle A,B\rangle}} \wedge A(x) \to Q_B(y)\}$
8: $\mathcal{P} :=$ Compile-Horn$(\mathcal{D}_\vee \cup \Omega(\mathcal{D}))$
9: $\mathcal{P} := \mathcal{P} \cup \Xi(\mathcal{D})$
10: **for each** unary predicate A in $\mathcal{D}$, replace Q_A with A in $\mathcal{P}$
11: **return** $\mathcal{P}$

- in each rule $N_{e_{\langle A,B\rangle}}(x, y) \wedge A(x) \to B(y)$ in $\mathcal{D}'$, we replace predicate B with a fresh predicate Q_B, which ensures acyclicity of the resulting program; and
- we apply Procedure 1 only to the monadic rules in $\mathcal{D}'$, thus ignoring programs $\mathcal{P}_{\langle A,B\rangle}$ for the purpose of resolution.

The algorithm based on these ideas is given by Procedure 3. In Lines (1)-(7), the procedure computes the following intermediate programs, where A, B range over the unary predicates in the input:

$$\Omega(\mathcal{D}) = \bigcup_{\langle A,B\rangle} \{N_{e_{\langle A,B\rangle}}(x, y) \wedge A(x) \to Q_B(y)\} \qquad \Xi(\mathcal{D}) = \bigcup_{\langle A,B\rangle} \mathcal{P}_{e_{\langle A,B\rangle}}$$

In Line (8), our algorithm invokes Procedure 1 to compute the Horn consequences of $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$. The following lemma establishes termination of this step.

Lemma 4.12. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program that is separable. Then, Procedure 1 terminates on $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$.*

The following lemma shows that no Horn consequence is lost by applying Procedure 1 to $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$ only, and appending the non-monadic program $\Xi(\mathcal{D})$ only afterwards.

Lemma 4.13. *Let $\mathcal{D}$ be a separable $\mathcal{ELU}$-program. Let $\mathcal{P}$ be the union of $\Xi(\mathcal{D})$ and the output of Procedure 1 on $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$. Then, for every query $A(x)$ and every ABox $\mathcal{A}$, we have* $\mathsf{cert}(A(x), \mathcal{D}, \mathcal{A}) = \mathsf{cert}(Q_A(x), \mathcal{P}, \mathcal{A})$.

Finally, in order to obtain a proper rewriting, in Line (10) we eliminate the auxiliary unary predicates Q_A before returning the output.

Theorem 4.14. *On input a separable $\mathcal{ELU}$-program $\mathcal{D}$, Procedure 3 returns a Datalog rewriting of $\mathcal{D}$.*

Example 4.15 *For our example program $\mathcal{D}^2$, we have the following:*

$$\Xi(\mathcal{D}^2) = \bigcup_{\langle A,B\rangle} P_{e_{\langle A,B\rangle}} = \mathcal{P}_{R(RR)^*} = \{(1),(2),(3),(4),(5)\}$$
$$\Omega(\mathcal{D}^2) = \{N_{R(RR)^*}(x,y) \wedge A(x) \rightarrow Q_B(y); \quad N_{R(RR)^*}(x,y) \wedge B(x) \rightarrow Q_A(y);$$
$$N_{(RR)^*}(x,y) \wedge A(x) \rightarrow Q_A(y); \quad N_{(RR)^*}(x,y) \wedge B(x) \rightarrow Q_B(y)\}$$

When applied to $\mathcal{D}^2_\vee \cup \Omega(\mathcal{D}^2)$, Procedure 1 computes the following additional Datalog rules:

$$N_{R(RR)^*}(x,y) \wedge N_{(RR)^*}(x,y) \rightarrow Q_A(y) \tag{6}$$
$$N_{R(RR)^*}(x,y) \wedge N_{(RR)^*}(x,y) \rightarrow Q_B(y) \tag{7}$$

Procedure 3 finally returns a Datalog program $\mathcal{P}$ consisting of rules $\Xi(\mathcal{D}^2)$, $\Omega(\mathcal{D}^2)$, as well as rules (6)-(7) *with predicates Q_A and Q_B replaced with A and B, respectively. Program $\mathcal{P}$ is a rewriting of $\mathcal{D}^2$.*

5 Discussion and Future Work

In this paper, we have presented two sufficient conditions for FO and Datalog rewritability of instance queries w.r.t. $\mathcal{ELU}$-ontologies. Our results are still rather preliminary, and we are currently working on extending them in several ways.

First, we are confident that our sufficient conditions can be extended to cover also non-linear $\mathcal{ELU}$-programs, as well as more expressive DLs such as $\mathcal{ALCHI}$. Devising a resolution-based rewriting algorithm in such extended setting becomes, however, a much more challenging task.

Example 5.1 *Consider the nonlinear $\mathcal{ELU}$-program $\mathcal{D}^3$ with the following rules:*

$$\top(x) \rightarrow A(x) \vee B(x) \qquad R(x,y) \wedge A(y) \rightarrow D(x) \qquad B(x) \rightarrow A(x)$$
$$D(x) \rightarrow H_1(x) \vee H_2(x) \qquad D(x) \wedge E(x) \rightarrow H(x) \qquad D(x) \rightarrow E(x)$$

This program is FO-rewritable: the non-linear rule $D(x) \wedge E(x) \rightarrow H(x)$ can be replaced with $D(x) \rightarrow H(x)$ to obtain an equivalent linear program that satisfies our sufficient condition in Section 4.1. Procedure 1, however, does not terminate when given $\mathcal{D}^3$ as input; in particular, Procedure 1 will compute the following infinite family of disjunctive rules for each $n > 1$:

$$\left[\bigwedge_{k=1}^{n} R(y_k,x_k) \wedge R(y_{k-1},x_k)\right] \wedge D(y_n) \rightarrow \left[\bigvee_{k=1}^{n} H(y_k)\right] \vee H_1(y_0) \vee H_2(y_0)$$

Second, it would be interesting to devise a resolution-based decision procedure for FO and Datalog rewritability for $\mathcal{ALC}$ ontologies; this is possible, since decidability has been recently shown. Finally, we are planning to implement our resolution algorithms and test them in practice.

References

1. Acciarri, A., Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QuOnto: Querying ontologies. In: AAAI. pp. 1670–1671 (2005)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. Artif. Intell. Res. 36, 1–69 (2009)
3. Bienvenu, M., Lutz, C., Wolter, F.: Deciding FO-rewritability in EL. In: DL. CEUR Workshop Proceedings, vol. 846 (2012)
4. Bienvenu, M., ten Cate, B., Lutz, C., Wolter, F.: Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. In: ACM PODS (2013), arXiv:1301.6479
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. Autom. Reasoning 39(3), 385–429 (2007)
6. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting in OWL 2 QL. In: CADE. pp. 192–206 (2011)
7. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs: Preliminary report. In: Simon, J. (ed.) STOC '88. pp. 477–490. ACM (1988)
8. Cuenca Grau, B., Motik, B., Stoilos, G., Horrocks, I.: Computing datalog rewritings beyond Horn ontologies. In: IJCAI (2013), arXiv:1304.1402
9. Del Val, A.: First order LUB approximations: Characterization and algorithms. Artif. Intell. 162(1-2), 7–48 (2005)
10. Eiter, T., Ortiz, M., Simkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: AAAI (2012)
11. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: IJCAI. pp. 466–471 (2005)
12. Hustadt, U., Motik, B., Sattler, U.: Reasoning in Description Logics by a Reduction to Disjunctive Datalog. J. Autom. Reasoning 39(3), 351–384 (2007)
13. Naughton, J.F.: Data independent recursion in deductive databases. In: Silberschatz, A. (ed.) PODS '86. pp. 267–279. ACM (1986)
14. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. Appl. Log. 8(2), 186–209 (2010)
15. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over DL-Lite ontologies. In: KR (2012)
16. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: KR (2010)
17. Rossman, B.: Homomorphism preservation theorems. J. ACM 55(3) (2008)
18. Selman, B., Kautz, H.: Knowledge compilation and theory approximation. J. ACM 43(2), 193–224 (1996)
19. Stocker, M., Smith, M.: Owlgres: A scalable OWL reasoner. In: OWLED (2008)

A Proofs for Section 4.1 (FO Rewritability)

Given a disjunctive program $\mathcal{D}$, we write $\Sigma(\mathcal{D})$ for the set of all predicate symbols in $\mathcal{D}$. We call $\Sigma(\mathcal{D})$ the *signature* of $\mathcal{D}$. If $\mathcal{D}$ is an $\mathcal{ELU}$-program, we partition $\Sigma(\mathcal{D})$ into $\Sigma_c(\mathcal{D})$, the set of all unary predicates (concept names) in $\mathcal{D}$, and $\Sigma_r(\mathcal{D})$, the set of all binary predicates (role names) in $\mathcal{D}$.

We write $\mathsf{Clo}_R(\mathcal{D})$ for the set of all rules derivable from $\mathcal{D}$ by binary resolution, $\mathsf{Clo}_F(\mathcal{D})$ for the set of all rules derivable from $\mathcal{D}$ by positive factoring, and $\mathsf{Clo}_{RF}(\mathcal{D})$ for the set of all rules derivable from $\mathcal{D}$ by binary resolution and positive factoring.

Let $\boldsymbol{x}$ be a sequence of variables and let φ be a formula. We write $\boldsymbol{x}|_z^y$ and $\varphi|_z^y$ for the result of substituting y for z in $\boldsymbol{x}$ and φ, respectively. We write $\mathrm{FV}(\varphi)$ for the set of variables that occur free in φ. Since we only consider formulas that are conjunctions or disjunctions of atoms, we can also view them as sets of atoms, writing $A \in \varphi$ for "A occurs in φ". The notation $A \in r$ for a rule r is defined analogously.

Proposition A.1. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program. Every rule in $\mathsf{Clo}_R(\mathcal{D})$ has the form $A(x_0) \wedge \varphi \to \bigvee_{i=1}^n B_i(x_i)$ where φ is a conjunction of binary atoms such that $\{x_0, \dots, x_n\} \subseteq \mathrm{FV}(\varphi)$.*

Given a rule r of the form $A(x_0) \wedge \varphi \to \bigvee_{i=1}^n B_i(x_i)$, where φ is a conjunction of binary atoms such that $\{x_0, \dots, x_n\} \subseteq \mathrm{FV}(\varphi)$, we define:

$$G_r := (\mathrm{FV}(\varphi), \{\, (x, y) \mid \exists R\colon R(y, x) \in \varphi \,\})$$

We call r *tree-shaped* if the following conditions are satisfied:

1. G_r is a tree with x_0 as its root and $x_1, \dots, x_n$ as its leaves.
2. If $\{R_1(x, y), R_2(x, y)\} \subseteq \varphi$, then $R_1 = R_2$.

The *depth* of r is defined as the depth of G_r.

Lemma A.2. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program. Every rule in $\mathsf{Clo}_R(\mathcal{D})$ is tree-shaped.*

Proof. By induction on the derivation of $r \in \mathsf{Clo}_R(\mathcal{D})$. Clearly, all rules in $\mathcal{D}$ are tree-shaped. So, w.l.o.g., let r be a resolvent or $r_1 = A(x) \wedge \varphi_1 \to \bigvee_{i=1}^m B_i(x_i)$ and $r_2 = B_1(y) \wedge \varphi_2 \to \bigvee_{i=1}^n C_i(y_i)$ on $B_1(x_1)$ and $B_1(y)$, respectively. By the inductive hypothesis, r_1 and r_2 are tree-shaped. Since $\mathrm{FV}(\varphi_1)$ and $\mathrm{FV}(\varphi_2)$ are assumed to be disjoint and the tree corresponding to r_2 is rooted at y, the graph $(\mathrm{FV}(\varphi_1 \wedge (\varphi_2|_y^{x_1})), \{\, (x, y) \mid \exists R\colon R(y, x) \in \varphi_1 \wedge (\varphi_2|_y^{x_1}) \,\})$ is a tree rooted at x that has $x_2, \dots, x_m, y_1, \dots, y_n$, as its leaves. Moreover, since y has no incoming edge in G_{r_2}, we have $R(x_1, z) \in \varphi_1 \wedge (\varphi_2|_y^{x_1})$ if and only if $R(x_1, z) \in \varphi_1$ (for every variable z). The claim follows. □

Lemma A.3. *Let $\mathcal{D}$ be a linear, acyclic $\mathcal{ELU}$-program and let n be the maximal transfer depth of a unary predicate in $\mathcal{D}$. Then each rule in $\mathsf{Clo}_R(\mathcal{D})$ has depth at most n.*

Proof. Let $r = A(x) \wedge \varphi \to \bigvee_{i=1}^{n} B_i(x_i) \in \mathsf{Clo}_R(\mathcal{D})$ By Lemma A.2, it suffices to show that, for every $i \in [1, n]$, the length of the (unique) path from x to x_i in G_r is equal to the difference between the transfer depth of A and the transfer depth of B_i. This follows by a straightforward induction on the derivation of r from $\mathcal{D}$. □

Definition A.4 (n-Simulation). *Let $r = A(x_0) \wedge \varphi \to \psi$ be a tree-shaped rule. We define an inductive family of preorders $(\preceq_r^n)_{n \in \mathbb{N}}$ between variables in r as follows:*

$$\begin{aligned} x \preceq_r^0 y \;&:\Leftrightarrow\; \{\, B \mid B(x) \in \psi \,\} \subseteq \{\, B \mid B(y) \in \psi \,\} \\ x \preceq_r^{n+1} y \;&:\Leftrightarrow\; x \preceq_r^0 y \text{ and for every } R \text{ and } x' \text{ such that } R(x', x) \in \varphi \\ &\qquad \text{there is some } y' \text{ such that } R(y', y) \in \varphi \text{ and } x' \preceq_r^n y' \end{aligned}$$

*We say x is n-*simulated *by y in r if $x \preceq_r^n y$. We define $[x]_r^n := \{\, y \mid x \preceq_r^n y$ and $y \preceq_r^n x \,\}$.*

Lemma A.5. *Let r be a tree-shaped rule of depth m and let x, y, z be variables such that, for some R, $R(x, z), R(y, z) \in r$ and $x \preceq_r^n y$ for some $n \geq m$. There is a substitution σ such that $x\sigma = y$, $r\sigma$ is tree-shaped, and $r\sigma \subseteq r$.*

Proof. The claim follows if we can show that for every tree-shaped rule r of depth m and every pair x, y such that $x \preceq_r^m y$, there is a substitution σ such that $x\sigma = y$ and for every pair u, v of variables in the subtree of G_r rooted at x we have:

1. $u\sigma, v\sigma$ are variables in the subtree of G_r rooted at y.
2. For every variable u not in the subtree of G_r rooted at x: $u\sigma = u$.
3. For every unary predicate B: if $B(u) \in r$, then $B(u)\sigma \in r$.
4. For every binary predicate R': if $R'(u, v) \in r$, then $R'(u, v)\sigma \in r$.

We prove the claim by induction on m. If $m = 0$, x and y have no successors in G_r. Therefore, it suffices to show (1-3). Let σ map x to y and every other variable in r to itself. Thus, (1) and (2) are trivial. For (3), observe that, since $x \preceq_r^0 y$, we have $B(x)\sigma = B(y) \in r$ whenever $B(x) \in r$.

Now suppose $m > 0$. Let $R_1, \ldots, R_n, x_1, \ldots, x_n$ be all the predicate symbols and variables such that $R_i(x_i, x) \in r$ ($i \in [1, n]$). Since $x \preceq_r^m y$, there are $y_1, \ldots, y_n$ such that $R_i(y_i, y) \in r$ and $x \preceq_r^{m-1} y$. By the inductive hypothesis, there are substitutions $\sigma_1, \ldots, \sigma_n$ such that, for every $i \in [1, n]$: $x_i\sigma_i = y_i$ and σ_i satisfies (1-4) for the subtree of G_r rooted at x_i. Let σ be defined such that:

- $x\sigma = y$.
- For every $i \in [1, n]$ and every variable u in the subtree of x_i: $u\sigma = u\sigma_i$.
- For every other variable u: $u\sigma = u$.

Note that σ is well-defined since the subtrees rooted at $x_1, \ldots, x_n$ are pairwise disjoint and do not contain x (which, in turn, holds since r is tree-shaped). Clearly, σ satisfies (1) and (2). For the subtrees of G_r rooted at $x_1, \ldots, x_n$, (3) and (4) hold since σ extends $\sigma_1, \ldots, \sigma_n$. For x, (3) holds since $x \preceq_r^0 y$, and (4) follows since we have $R_i(x, x_i)\sigma = R_i(y, y_i)$ for every $i \in [1, n]$. □

Proposition A.6. *Let $\mathcal{D}$ be a monadic disjunctive program and let $f : \mathbb{N} \to \mathbb{N}$ be a function defined as follows:*

$$\begin{aligned} f(0) &:= 2^{|\Sigma_c(\mathcal{D})|} \\ f(n+1) &:= 2^{|\Sigma_c(\mathcal{D})|} \cdot |\Sigma_r(\mathcal{D})| \cdot 2^{f(n)} \end{aligned}$$

Then $[x]_r^n \leq f(n)$ for every rule r over $\Sigma(\mathcal{D})$ and every variable x in r.

A rule r is *condensed* if r has no condensation that is smaller than r. By Proposition A.6 and Lemma A.5 we obtain:

Lemma A.7. *Let $\mathcal{D}$ be an $\mathcal{ELU}$-program. The size of condensed tree-shaped rules of depth n over $\Sigma(\mathcal{D})$ is bounded in n and $|\Sigma(\mathcal{D})|$.*

Proof. Let $\mathcal{D}$ be an $\mathcal{ELU}$-program and let f be defined as in Proposition A.6. It suffices to show that the number of variables in every condensed tree-shaped rule of depth n is bounded by $n \cdot (|\Sigma_r(\mathcal{D})| \cdot f(n))^n$.

Note that every tree T of depth n contains at most $n \cdot m^n$ nodes, where m is the maximal outdegree of a node in T. Therefore, every tree of depth n that has k nodes must contain a node that has outdegree at least $\sqrt[n]{k/n}$.

Suppose, for contradiction, r is a condensed tree-shaped rule of depth n that mentions more than $n \cdot (|\Sigma_r(\mathcal{D})| \cdot f(n))^n$ variables. By the above observation, G_r must have a node z that has outdegree greater $|\Sigma_r(\mathcal{D})| \cdot f(n)$. Consequently, by Proposition A.6, there is some predicate R and two distinct variables x, y such that $R(x, z), R(y, z) \in r$ and $[x]_r^n = [y]_r^n$. By Lemma A.5, there is a substitution σ such that $x\sigma = y$ and $r\sigma \subseteq r$. Since x and y are distinct, it follows that $r\sigma \subsetneq r$, contradicting the assumption that r is condensed. □

Lemma A.8. *Every condensation of a tree-shaped rule is tree-shaped.*

Proof. Let r be a tree-shaped rule and $r\sigma$ a condensation of r. Since $r\sigma$ is a subclause of r, $G_{r\sigma}$ must be a forest, the root of G_r must be a source in $G_{r\sigma}$, and all leaves of G_r must be sinks in $G_{r\sigma}$. Since $r\sigma$ is a substitution instance of r and G_r is connected, so must be $G_{r\sigma}$. Hence, $G_{r\sigma}$ is a tree. The claim follows. □

Lemma A.9. *Unrestricted binary resolution with condensation terminates on linear, acyclic $\mathcal{ELU}$-programs. Moreover, the size of every rule derivable by binary resolution with condensation from an acyclic $\mathcal{ELU}$-program $\mathcal{D}$ is bounded in the size of $\mathcal{D}$.*

Proof. Let $\mathcal{D}$ be a linear, acyclic $\mathcal{ELU}$-program. By Lemma A.2, Lemma A.5, and Lemma A.8, every rule derivable by resolution with condensation is tree-shaped. By Lemma A.3 and Lemma A.7, the size and hence the number of condensed tree-shaped rules is bounded in $|\Sigma(\mathcal{D})|$. The claims follow. □

Lemma A.10 ([8] Theorem 12, Lemma 19). *Let $\mathcal{D}$ be a disjunctive program, let $\mathcal{A}$ be an ABox, and let C be a Horn clause such that $\mathcal{D} \cup \mathcal{A} \models C$. Let $\mathcal{D}_H^i$ be the Datalog program computed by Procedure 1 after i iterations of the main loop. Then there is some n such that $\mathcal{D}_H^n \cup \mathcal{A} \models C$.*

Theorem A.11. *Let $\mathcal{D}$ be a linear, acyclic $\mathcal{ELU}$-program. Procedure 1 terminates on $\mathcal{D}$ and returns a Datalog rewriting of $\mathcal{D}$.*

Proof. To show termination of Procedure 1, it suffices to bound the size of every rule derived from $\mathcal{D}$ by resolution with condensation and factoring. By Lemma A.9, the size of every rule derived from $\mathcal{D}$ by resolution with condensation but no factoring is bounded in $|\Sigma(\mathcal{D})|$. Thus, it suffices to show that for every rule r obtained from $\mathcal{D}$ by resolution with condensation and factoring there is a (not necessarily strictly) larger rule r' obtained from $\mathcal{D}$ by resolution with condensation but without factoring such that r is subsumed by r' (where clauses are viewed as sets of atoms). This follows by a straightforward induction on the derivation of r.

By Lemma A.10, the output of Procedure 1 is a Datalog rewriting of $\mathcal{D}$. □

Let $\mathcal{D}$ be a linear, acyclic $\mathcal{ELU}$-program and let $r = A(x) \land \varphi \to \bigvee_{i=1}^{n} B_i(x_i) \in \mathsf{Clo}_R(\mathcal{D})$. The proof of Lemma A.3 exhibits that the length of the path from x to x_i in G_r is equal to the difference between the transfer depth of A and the transfer depth of B_i. In particular, if $B_i = B_j$ for some $1 \leq i < j \leq n$, then x_i and $_j$ must have the same distance from x in G_r. Thus, while factoring can destroy the tree structure of G_r, it does not change the distance between x and x_i in G_r. The same is true for condensation. Thus, we have:

Proposition A.12. *Let $\mathcal{D}$ be a linear, acyclic $\mathcal{ELU}$-program, let $\mathcal{P}$ be a Datalog rewriting of $\mathcal{D}$ computed by Procedure 1, and let $r = A(x) \land \varphi \to \bigvee_{i=1}^{n} B_i(x_i) \in \mathcal{P}$. Then for every $i \in [1, n]$, the distance between x and x_i in G_r is equal to the difference between the transfer depth of A and the transfer depth of B_i.*

Thus, given an $\mathcal{ELU}$-program $\mathcal{D}$, its Datalog rewriting $\mathcal{P}$, and a rule $r \in \mathcal{P}$, the depth of G_r is still bounded by the maximal transfer depth of a predicate in $\mathcal{D}$ and cannot be further increased by resolution.

Theorem 4.4. *Given an acyclic $\mathcal{ELU}$-program $\mathcal{D}$ and a unary predicate A in $\mathcal{D}$, Procedure 2 terminates and returns a UCQ rewriting of $A(x)$ w.r.t. $\mathcal{D}$.*

Proof. By Theorem A.11, the call to Procedure 1 (Compile-Horn) terminates and returns a Datalog rewriting of $\mathcal{D}$. It is easily seen that the main loop of the procedure computes the *A-expansions* of $\mathcal{D}$ as defined in [7] (similarly to the algorithm in [13]). Therefore, the procedure terminates and returns a UCQ-rewriting of $A(x)$ w.r.t. $\mathcal{D}$ provided the resolution strategy implemented in the main loop terminates.

Suppose, for contradiction, this is not the case. Then for every $n \in \mathbb{N}$, $\mathcal{P}''$ must eventually contain a rule r such that G_r contains more than n nodes and r is not subsumed by any clause previously added to $\mathcal{P}''$. By Proposition A.12, the depth of every rule in $\mathsf{Clo}_R(\mathcal{P} \cup \{A(x) \to Q(x)\})$ is bounded in $\mathcal{D}$. So, let d be the corresponding bound. It follows that G_r must contain a variable x with outdegree at least $\sqrt[d]{n/d}$. For large enough n, this means there is a rule $r \in \mathcal{P}$ and rules $r_1 = B(y) \land \varphi_1 \to Q(x)$, $r_2 = B(y) \land \varphi_2 \to Q(x) \in \mathcal{P}'$, r_1', r_2' such that r_1' is a resolvent of r with r_1, r_2 is derived from r_1' (and hence $\varphi_1 \subseteq \varphi_2$), and

r'_2 is a resolvent of r with r_2. It is easily seen that r'_1 subsumes r'_2. Moreover, since r'_2 is derived from r'_1, r'_1 is added to $\mathcal{P}''$ before r'_2, in contradiction to our assumption. □

B Proofs for Section 4.2 (Datalog Rewritability)

Given a separable $\mathcal{ELU}$-program $\mathcal{D}$, we write $\mathcal{D}_\vee$ for the fragment of $\mathcal{D}$ as used in Definition 4.6 and $\mathcal{D}_\triangledown$ for $\mathcal{D} \setminus \mathcal{D}_\vee$.

Lemma 4.12. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program that is separable. Then, Procedure 1 terminates on $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$.*

Proof. The claim holds since $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$ is acyclic, which is the case since $\mathcal{D}_\vee$ is acyclic and every rule added by $\Omega(\mathcal{D})$ leads to a sink. □

For the proof of Lemma 4.13, we need to generalise the notion of annotations from how they are defined in §4.2. Let $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ be finite sequences of variables. *Annotations* are now triples of the form $(\boldsymbol{x}, S, \boldsymbol{y})$, where S is a set of binary atoms whose free variables are contained in $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$. Let $\alpha = (\boldsymbol{x}, S, \boldsymbol{y})$ be an annotation and let $\boldsymbol{A} = A_1, \ldots, A_{|\boldsymbol{x}|}$, and $\boldsymbol{B} = B_1, \ldots, B_{|\boldsymbol{y}|}$ be finite sequences of unary predicates. We define the following notation for disjunctive rules:

$$\boldsymbol{A}\alpha\boldsymbol{B} := \forall \boldsymbol{x}\boldsymbol{y}\boldsymbol{z}. \left(\bigwedge_{i=1}^{|\boldsymbol{x}|} A_i(x_i) \right) \wedge \left(\bigwedge_{C \in S} C \right) \rightarrow \bigvee_{j=1}^{|\boldsymbol{y}|} B_j(y_j)$$

For linear rules, the notation reduces to $A\alpha\boldsymbol{B}$, while for Datalog rules, we have $\boldsymbol{A}\alpha B$, where A, B are single predicate symbols. If $A\alpha\boldsymbol{B}$ is an $\mathcal{ELU}$-rule, α must be of the form $(x, \{R(y, x)\}, y)$ or $(x, \emptyset, \underbrace{x \ldots x}_{n})$ for some $n \geq 1$.

Let $\alpha = (\boldsymbol{x}, S, \boldsymbol{y})$, $\beta = (\boldsymbol{x}', S', \boldsymbol{y}')$ be annotations such that $\mathrm{FV}(\alpha) \cap \mathrm{FV}(\beta) = \emptyset$, and let $i \in [1, |\boldsymbol{y}|]$, $j \in [1, |\boldsymbol{x}'|]$. We define:

$$\alpha \circ^i_j \beta := (\boldsymbol{x} x'_1 \ldots x'_{j-1} x'_{j+1} \ldots x'_{|\boldsymbol{x}'|}, S \cup (S'|^{y_i}_{x'_j}), y_1 \ldots y_{i-1} y_{i+1} \ldots y_{|\boldsymbol{y}|} (\boldsymbol{y}'|^{y_i}_{x'_j}))$$

We call $\alpha \circ^i_j \beta$ the *composition* of α at position i to β at position j. We abbreviate $\alpha \circ^1_j \beta$ as $\alpha \circ_j \beta$, $\alpha \circ^i_1 \beta$ as $\alpha \circ^i \beta$, and $\alpha \circ^1_1 \beta$ as $\alpha \circ \beta$.

Proposition B.1. *Let $\alpha = (\boldsymbol{x}_\alpha, S_\alpha, \boldsymbol{y}_\alpha), \beta = (\boldsymbol{x}_\beta, S_\beta, \boldsymbol{y}_\beta), \gamma = (\boldsymbol{x}_\gamma, S_\gamma, \boldsymbol{y}_\gamma)$ be annotations and let $i, i' \in [1, |\boldsymbol{y}_\alpha|]$, $j \in [1, |\boldsymbol{x}_\beta|]$, $k \in [1, |\boldsymbol{y}_\beta|]$, $l \in [1, |\boldsymbol{x}_\gamma|]$. Then:*

1. $\alpha \circ^i_j (\beta \circ^k_l \gamma) = (\alpha \circ^i_j \beta) \circ^{k+|\boldsymbol{y}_\alpha|-1}_l \gamma$.
2. *If $i < i'$, then* $(\alpha \circ^i_j \beta) \circ^{i'-1}_l \gamma \equiv_a (\alpha \circ^{i'}_l \gamma) \circ^i_j \beta$.

Let $r = \boldsymbol{A}\alpha\boldsymbol{B}$, $r' = \boldsymbol{A}'\alpha'\boldsymbol{B}'$ be two rules such that $B_i = A'_j$ and $\mathrm{FV}(\alpha) \cap \mathrm{FV}(\beta) = \emptyset$. We write $r +^i_j r'$ for the resolvent of r and r' on $B_i(y_i)$ and $A'_j(x'_j)$, respectively. We abbreviate $r +^1_j r'$ as $r +_j r'$, $r +^i_1 r'$ as $r +^i r'$, and $r +^1_1 r'$ as $r + r'$. Resolution naturally corresponds to composition of annotations:

Proposition B.2. *Let $r = A\alpha\boldsymbol{B}$, $r' = A'\alpha'\boldsymbol{B}'$ be linear rules such that $B_i = A'$ and $\mathrm{FV}(\alpha) \cap \mathrm{FV}(\alpha') = \emptyset$. Then:*

1. $r +^i r' = A(\alpha \circ^i \alpha')B_1 \dots B_{i-1}B_{i+1} \dots B_{|\boldsymbol{B}|}\boldsymbol{B}'$.
2. *If r is Datalog, then $r + r' = A(\alpha \circ \alpha')\boldsymbol{B}'$.*

We consider two rules r, r' to be the same (and write $r = r'$) if they are identical up to renaming of bound variables and reordering of literals. The corresponding equivalence on annotations can be defined as the least equivalence relation $\equiv_a$ such that:

1. $(x_1 \dots x_m, S, y_1 \dots y_n) \equiv_a (x_1 \dots x_{i-1}x_j x_{i+1} \dots x_{j-1}x_i x_{j+1} \dots x_m, S,$
   $y_1 \dots y_{k-1}y_l y_{k+1} \dots y_{l-1}y_k y_{l+1} \dots y_n)$
   for all $1 \leq i \leq j \leq m$ and $1 \leq k \leq l \leq n$,
2. $(\boldsymbol{x}, S, \boldsymbol{y}) \equiv_a (\boldsymbol{x}|_v^u, S|_v^u, \boldsymbol{y}|_v^u)$ for every $u \notin \boldsymbol{x} \cup \boldsymbol{y} \cup \mathrm{FV}(S)$.

Proposition B.3. *Two rules of the form $\boldsymbol{A}\alpha\boldsymbol{B}$, $\boldsymbol{A}\beta\boldsymbol{B}$ are identical up to renaming of bound variables and reordering of literals if and only if $\alpha \equiv_a \beta$.*

We assume $\circ_j^i$ and $+_j^i$ to be left-associative, writing $r_1 +_{j_1}^{i_1} r_2 +_{j_2}^{i_2} r_3$ for $(r_1 +_{j_1}^{i_1} r_2) +_{j_2}^{i_2} r_3$ and $\alpha_1 \circ_{j_1}^{i_1} \alpha_2 \circ_{j_2}^{i_2} \alpha_3$ for $(\alpha_1 \circ_{j_1}^{i_1} \alpha_2) \circ_{j_2}^{i_2} \alpha_3$. As corollaries of Proposition B.1, we obtain:

Corollary B.4. *Let $r = A\alpha\boldsymbol{B}$, r', r'' be linear rules and i, j indices such that $r +^i (r' +^j r'')$ is a resolvent of r, r' and r''. Then:*

1. $r +^i (r' +^j r'') = r +^i r' +^{j+|\boldsymbol{B}|-1} r''$.
2. *If r is Datalog, then $r + (r' +^j r'') = r + r' +^j r''$.*

Corollary B.5. *Let $r = A\alpha\boldsymbol{B}$, r', r'' be linear rules and i, j indices such that $i < j$ and $r +^j r'' +^i r'$ is a resolvent of r, r', and r''. Then $r +^i r' +^{j-1} r'' = r +^j r'' +^i r'$.*

Lemma B.6. *Let $\mathcal{D}$ be a separable $\mathcal{ELU}$-program, let $\mathcal{P}$ be a Datalog program such that $\Sigma(\mathcal{P}) \cap \Sigma_c(\mathcal{D}) = \emptyset$ and no rule in $\mathcal{P}$ resolves with a rule in $\mathcal{D}_\vee$, and let $r \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{P})$ be linear. Then either $r \in \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$ or, for some $n \geq 0$, we have $r = r_\vee + r_1 + \ldots + r_n$ where:*

- $r_\vee = A\alpha\boldsymbol{B} \in \mathsf{Clo}_R(\mathcal{D}_\vee)$ *where* $|\boldsymbol{B}| \geq n$.
- $\{r_1, \dots, r_n\} \subseteq \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$.

Proof. Let $r \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{P}) \setminus \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$ we show that r can be decomposed as claimed by induction on the derivation of $r \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{P})$. If $r \in \mathcal{D}$, the claim is true for $n = 0$. Otherwise, let $r = r' +_j^i r''$. Since $r \notin \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$, we have $\{r', r''\} \not\subseteq \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$. Therefore, since no rule in $\mathcal{P}$ resolves with $\mathcal{D}_\vee$ and no unary predicate in $\mathcal{D}_\triangledown$ occurs in the body of a rule in $\mathcal{D}_\vee$, we must have $r' \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{P}) \setminus \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$. Hence, by the inductive hypothesis, we have $r' = r'_\vee + r'_1 + \ldots + r'_m$ where $r'_\vee = A'\alpha'\boldsymbol{B}' \in \mathsf{Clo}_R(\mathcal{D}_\vee)$ and $\{r'_1, \dots, r'_m\} \subseteq \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$. Since r is linear and $\Sigma(\mathcal{P}) \cap \Sigma(\mathcal{D}_\vee) = \emptyset$, r'' must be linear, and hence $j = 1$. We distinguish two cases:

1. $r'' \in \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$. Then either $i \in [m{+}1, n]$ or $i \in [1, m]$. In the former case, the claim is immediate. In the latter case, the claim follows by Corollary B.4.
2. $r'' \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{P}) \setminus \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$. Then $r'' = r''_\vee + r''_1 + \ldots + r''_k$ where $r''_\vee = A''\alpha''\boldsymbol{B}'' \in \mathsf{Clo}_R(\mathcal{D}_\vee)$ and $\{r''_1, \ldots, r''_k\} \subseteq \mathsf{Clo}_R(\mathcal{D}_\triangledown \cup \mathcal{P})$. Moreover, $n = m + k$ an $|\boldsymbol{B}| = |\boldsymbol{B}'| + |\boldsymbol{B}''|$. Since no unary predicate in $\mathcal{D}_\triangledown$ occurs in the body of a rule in $\mathcal{D}_\vee$, we have $i \in [m+1, n]$. W.l.o.g., let $i = 1$ (we can always achieve this by reordering the literals in r''). Then

$$\begin{aligned} r &= r'_\vee + r'_1 + \ldots + r'_m + (r''_\vee + r''_1 + \ldots + r''_k) \\ &= r'_\vee + r'_1 + \ldots + r'_m + r''_\vee +^{|\boldsymbol{B}'|} r''_1 +^{|\boldsymbol{B}'|} \ldots +^{|\boldsymbol{B}'|} r''_k \\ &= (r'_\vee +^{m+1} r''_\vee) + r'_1 + \ldots + r'_m +^{|\boldsymbol{B}'|} r''_1 +^{|\boldsymbol{B}'|} \ldots +^{|\boldsymbol{B}'|} r''_k \end{aligned}$$

by Corollary B.4 and Corollary B.5. Therefore, we have $r = r_\vee + r'_1 + \ldots + r'_m + r''_1 + \ldots + r''_k$ where $r_\vee$ is obtained from $r'_\vee +^{m+1} r''_\vee$ by moving the last $|\boldsymbol{B}''|$ literals right after the first m literals. □

As shown in §4.2, every linear $\mathcal{EL}$-program $\mathcal{P}$ can be seen as a labeled transition system (i.e., automaton without initial and final states). Every rule $A\alpha B \in \mathcal{P}$ is seen as a transition from A to B labeled with α. This view yields a natural notion of a *path* between two unary predicates in $\mathcal{P}$ as a sequence of "connected" rules.

Proposition B.7. *Let $\mathcal{P}$ be a linear $\mathcal{EL}$-program and let r be a rule. Then $r \in \mathsf{Clo}_R(\mathcal{P})$ iff there is a path $r_1, \ldots, r_n$ $(n \geq 1)$ in $\mathcal{P}$ such that $r = r_1 + \ldots + r_n$.*

Corollary B.8. *Let $\mathcal{P}$ be a linear $\mathcal{EL}$-program and let $r = A\alpha B$ be a rule where $A \neq B$. Then $r \in \mathsf{Clo}_R(\mathcal{P})$ if and only if there is a word $\alpha_1 \ldots \alpha_n \in \mathcal{L}(\mathsf{Aut}_\mathcal{P}(A, B))$ such that $\alpha = \alpha_1 \circ \ldots \circ \alpha_n$.*

Given an annotation $\alpha = (u, S, v)$ and variables $x, y \notin \mathrm{FV}(S)$, we write $\alpha(x, y)$ for the formula $\bigwedge_{C \in S}(C|^{xy}_{uv})$. If S is empty, the notation is only defined if $x = y$ and stands for $\top(x)$.

With this, we can generalise the definition of the Datalog program $\mathcal{P}_e$ for a regular expression e over annotations (see §4.2) as follows:

$$\begin{aligned} \mathcal{P}_\emptyset &= \emptyset \\ \mathcal{P}_\alpha &= \{\alpha(x, y) \to N_\alpha(x, y)\} \\ \mathcal{P}_{e_1 e_2} &= \mathcal{P}_{e_1} \cup \mathcal{P}_{e_2} \cup \{N_{e_1}(x, y) \wedge N_{e_2}(y, z) \to N_{e_1 e_2}(x, z)\} \\ \mathcal{P}_{e_1 + e_2} &= \mathcal{P}_{e_1} \cup \mathcal{P}_{e_2} \cup \{N_{e_1}(x, y) \to N_{e_1+e_2}(x, y),\ N_{e_2}(x, y) \to N_{e_1+e_2}(x, y)\} \\ \mathcal{P}_{e^*} &= \mathcal{P}_e \cup \{\top(x) \to N_{e^*}(x, x),\ N_e(x, y) \wedge N_{e^*}(y, z) \to N_{e^*}(x, z)\} \end{aligned}$$

Proposition B.9. *Let $\mathcal{P}$ be a linear $\mathcal{EL}$-program, let e be a regular expression over annotations in $\mathcal{P}$, and let $\mathcal{L}(e)$ be the language represented by e. Then for every sequence $\alpha_1, \ldots, \alpha_n$ $(n \geq 1)$ of annotations in $\mathcal{P}$:*

$$\alpha_1 \ldots \alpha_n \in \mathcal{L}(e) \text{ iff } (\alpha_1 \circ \ldots \circ \alpha_n)(x, y) \to N_e(x, y) \in \mathsf{Clo}_R(\mathcal{P}_e)$$

Proposition B.10. *Let $\mathcal{P}$ be a linear $\mathcal{EL}$-program and let $e_1, \ldots, e_n$ be regular expressions over rule annotations in $\mathcal{P}$. Then $\mathcal{P}_{e_1} \cup \cdots \cup \mathcal{P}_{e_n}$ is a conservative extension of $\mathcal{P}_{e_1}$ in the sense that for every conjunction of atoms φ:*

$$\varphi \to N_{e_1}(x, y) \in \mathsf{Clo}_R(\mathcal{P}_{e_1}) \text{ iff } \varphi \to N_{e_1}(x, y) \in \mathsf{Clo}_R(\mathcal{P}_{e_1} \cup \cdots \cup \mathcal{P}_{e_n})$$

Given a disjunctive program $\mathcal{D}$, we write $\mathsf{Clo}_R^S(\mathcal{D})$, $\mathsf{Clo}_F^S(\mathcal{D})$, and $\mathsf{Clo}_{RF}^S(\mathcal{D})$ for the set of all rules subsumed in $\mathsf{Clo}_R(\mathcal{D})$, $\mathsf{Clo}_F(\mathcal{D})$, and $\mathsf{Clo}_{RF}(\mathcal{D})$, respectively.

Lemma B.11. *Let $\mathcal{D}$ be an $\mathcal{ELU}$-program such that $\mathsf{Clo}_R(\mathcal{D}) = \mathcal{D}$ and let $\mathcal{R}$ be a Datalog program such that $\Sigma_c(\mathcal{D}) \cap \Sigma(\mathcal{R}) = \emptyset$. Then:*

$$\mathsf{Clo}_R^S(\mathcal{D} \cup \mathcal{R}) = \mathsf{Clo}_R^S(\mathcal{R}) \cup \left\{ \psi \wedge \varphi \to \chi \;\middle|\; \begin{array}{l} \psi \wedge \varphi' \to \chi \text{ is subsumed in } \mathcal{D}, \\ \varphi \to \varphi' \in \mathsf{Clo}_R^S(\mathcal{R}) \end{array} \right\}$$

where ψ denotes conjunctions of unary atoms, χ denotes disjunctions of unary atoms, and φ, φ' denote conjunctions of binary atoms.

Proof. The inclusion from left to right is immediate. For the other inclusion, suppose $r = \psi \wedge \varphi \to \chi \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{R}) \setminus \mathsf{Clo}_R(\mathcal{R})$. We show the existence of a rule $\psi \wedge \varphi' \to \chi \in \mathcal{D}$ such that $\varphi \to \varphi' \in \mathsf{Clo}_R^S(\mathcal{R})$ by induction on the derivation of $r \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{R})$. If $r \in \mathcal{D}$, the claim is immediate, so suppose r is obtained by resolution from $r_1, r_2 \in \mathsf{Clo}_R(\mathcal{D} \cup \mathcal{R})$. Since $r \notin \mathsf{Clo}_R(\mathcal{R})$, $\{r_1, r_2\} \not\subseteq \mathsf{Clo}_R(\mathcal{R})$. We prove the claim for $\{r_1, r_2\} \cap \mathsf{Clo}_R(\mathcal{R}) = \emptyset$ (the other case is similar but simpler). Let $r_1 = \psi_1 \wedge \varphi_1 \to \chi_1$ and $r_2 = \psi_2 \wedge \varphi_2 \to \chi_2$. Since r is a resolvent of r_1 and r_2, we have $\varphi = \varphi_1 \wedge \varphi_2$. By the inductive hypothesis, there are rules $r_1' = \psi_1 \wedge \varphi_1' \to \chi_1$ and $r_2' = \psi_2 \wedge \varphi_2' \to \chi_2$ such that $\varphi_1 \to \varphi_1' \in \mathsf{Clo}_R^S(\mathcal{R})$ and $\varphi_2 \to \varphi_2' \in \mathsf{Clo}_R^S(\mathcal{R})$. Then $r' = \psi \wedge \varphi_1' \wedge \varphi_2' \to \chi$ is a resolvent of r_1' and r_2' and hence r' is subsumed by a clause in $\mathcal{D}$. Since $\varphi_1 \to \varphi_1' \in \mathsf{Clo}_R^S(\mathcal{R})$ and $\varphi_2 \to \varphi_2' \in \mathsf{Clo}_R^S(\mathcal{R})$, we have $\varphi_1 \wedge \varphi_2 \to \varphi_1' \wedge \varphi_2' \in \mathsf{Clo}_R^S(\mathcal{R})$. The claim follows. □

Similarly, we have:

Lemma B.12. *Let $\mathcal{D}$ be an $\mathcal{ELU}$-program such that $\mathsf{Clo}_{RF}(\mathcal{D}) = \mathcal{D}$ and let $\mathcal{R}$ be a Datalog program such that $\Sigma_c(\mathcal{D}) \cap \Sigma(\mathcal{R}) = \emptyset$. Then:*

$$\mathsf{Clo}_{RF}^S(\mathcal{D} \cup \mathcal{R}) = \mathsf{Clo}_R^S(\mathcal{R}) \cup \left\{ \psi \wedge \varphi \to \chi \;\middle|\; \begin{array}{l} \psi \wedge \varphi' \to \chi \text{ is subsumed in } \mathcal{D}, \\ \varphi \to \varphi' \in \mathsf{Clo}_R^S(\mathcal{R}) \end{array} \right\}$$

Proof. Proceeds analogously to the proof of Lemma B.11 with the additional observation that $\mathsf{Clo}_R(\mathcal{R}) = \mathsf{Clo}_{RF}(\mathcal{R})$ since $\mathcal{R}$ is a Datalog program. □

Given a disjunctive program $\mathcal{D}$, we write $\mathcal{D}_H$ for the set of all Datalog (or Horn) rules in $\mathcal{D}$.

Lemma B.13. *Let $\mathcal{D}$ be an $\mathcal{ELU}$-program, let $\mathcal{R}$ be a Datalog program such that $\Sigma_c(\mathcal{D}) \cap \Sigma(\mathcal{R}) = \emptyset$, and let r be a Datalog rule. Then:*

$$r \in \mathsf{Clo}_{RF}^S(\mathcal{D} \cup \mathcal{R}) \text{ iff } r \in \mathsf{Clo}_R^S(\mathsf{Clo}_{RF}(\mathcal{D})_H \cup \mathcal{R})$$

Proof. The direction from left to right holds since $\mathsf{Clo}_R(\mathsf{Clo}_{RF}(\mathcal{D})_H \cup \mathcal{R}) \subseteq \mathsf{Clo}_{RF}(\mathcal{D} \cup \mathcal{R})$. For the other direction, suppose $r \in \mathsf{Clo}_{RF}(\mathcal{D} \cup \mathcal{R})$. Clearly, $\mathsf{Clo}_{RF}(\mathcal{D} \cup \mathcal{R}) \subseteq \mathsf{Clo}_{RF}(\mathsf{Clo}_{RF}(\mathcal{D}) \cup \mathcal{R})$. Therefore, by Lemma B.12, $r \in \mathsf{Clo}^S_R(\mathcal{R}) \cup \{\, \psi \wedge \varphi \to \chi \mid \psi \wedge \varphi' \to \chi \in \mathsf{Clo}^S_{RF}(\mathcal{D}),\ \varphi \to \varphi' \in \mathsf{Clo}^S_R(\mathcal{R}) \,\}$. On the other hand, since $\mathsf{Clo}_{RF}(\mathsf{Clo}_{RF}(\mathcal{D})_H) = \mathsf{Clo}_{RF}(\mathcal{D})_H$, we have $\mathsf{Clo}^S_R(\mathsf{Clo}_{RF}(\mathcal{D})_H \cup \mathcal{R}) = \mathsf{Clo}^S_R(\mathcal{R}) \cup \{\, \psi \wedge \varphi \to \chi \mid \psi \wedge \varphi' \to \chi \in \mathsf{Clo}^S_{RF}(\mathcal{D})_H,\ \varphi \to \varphi' \in \mathsf{Clo}^S_R(\mathcal{R}) \,\}$. The claim follows since r is Datalog. □

Lemma B.14. *Let $\mathcal{P}$ be a linear $\mathcal{EL}$-program, let $r = A \alpha B$ be a rule where $A \neq B$, and let $A \in S \subseteq \Sigma_c(\mathcal{P})$. Then:*

$$r \in \mathsf{Clo}^S_R(\mathcal{P}) \text{ iff } r \in \mathsf{Clo}^S_R(\{A(y) \wedge N_{e_{\langle A,B\rangle}}(x,y) \to B(x)\} \cup \mathcal{P}_{e_{\langle A,B\rangle}})$$

Proof. Let $r' = A(y) \wedge N_{e_{\langle A,B\rangle}}(x,y) \to B(x)$. For the direction from left to right, suppose $r \in \mathsf{Clo}_R(\mathcal{P})$. By Corollary B.8 and Proposition B.9, $r'' = \alpha(x,y) \to N_{e_{\langle A,B\rangle}}(x,y) \in \mathsf{Clo}_R(\mathcal{P}_{e_{\langle A,B\rangle}})$. The claim follows since r is a resolvent of r' and r''.

For the direction from right to left, suppose $r \in \mathsf{Clo}_R(\{r'\} \cup \mathcal{P}_{e_{\langle A,B\rangle}})$. Since $\mathsf{Clo}_{RF}(\{r'\}) = \{r'\}$ and $\Sigma(\mathcal{P}_{e_{\langle A,B\rangle}}) \cap \{A,B\} = \emptyset$, we have $r \in \{\, A(y) \wedge \varphi \to B(x) \mid \varphi \to N_{e_{\langle A,B\rangle}}(x,y) \in \mathsf{Clo}^S_R(\mathcal{P}_{e_{\langle A,B\rangle}}) \,\}$ (by Lemma B.12), and thus $r'' \in \mathsf{Clo}^S_R(\mathcal{P}_{e_{\langle A,B\rangle}})$. The claim follows by Proposition B.9 and Corollary B.8 (it is easily seen that the equivalence $A \alpha B \in \mathsf{Clo}_R(\mathcal{P}) \Leftrightarrow r'' \in \mathsf{Clo}_R(\mathcal{P}_{e_{\langle A,B\rangle}})$ implies $A \alpha B \in \mathsf{Clo}^S_R(\mathcal{P}) \Leftrightarrow r'' \in \mathsf{Clo}^S_R(\mathcal{P}_{e_{\langle A,B\rangle}})$). □

Proposition B.15. *Let r_1, r_2 be tree-shaped rules, r'_1 a positive factor of r_1, and r'_3 a resolvent of r'_1 and r_2. Then there is a rule r_3 such that:*

1. *r_3 can be obtained by resolution from r_1 and r_2 in at most two steps.*
2. *r'_3 is equivalent, up to subsumption, to a positive factor of r_3.*

Lemma B.16. *For every linear $\mathcal{ELU}$-program $\mathcal{D}$: $\mathsf{Clo}^S_{RF}(\mathcal{D}) = \mathsf{Clo}^S_F(\mathsf{Clo}_R(\mathcal{D}))$.*

Proof. Clearly, $\mathsf{Clo}_F(\mathsf{Clo}_R(\mathcal{D})) \subseteq \mathsf{Clo}_{RF}(\mathcal{D})$. The other inclusion follows with Proposition B.15 by induction on the derivation of $r \in \mathsf{Clo}_{RF}(\mathcal{D})$. □

Lemma B.17. *Let $\mathcal{D}$ be a linear $\mathcal{ELU}$-program and let $\mathcal{R}$ be a Datalog program such that $\Sigma_c(\mathcal{D}) \cap \Sigma(\mathcal{R}) = \emptyset$. Then $\mathsf{Clo}^S_{RF}(\mathcal{D} \cup \mathcal{R}) = \mathsf{Clo}^S_F(\mathsf{Clo}_R(\mathcal{D} \cup \mathcal{R}))$.*

Proof. Clearly, $\mathsf{Clo}_F(\mathsf{Clo}_R(\mathcal{D} \cup \mathcal{R})) \subseteq \mathsf{Clo}_{RF}(\mathcal{D} \cup \mathcal{R})$. So, let $r \in \mathsf{Clo}_{RF}(\mathcal{D} \cup \mathcal{R}) = \mathsf{Clo}_{RF}(\mathsf{Clo}_{RF}(\mathcal{D}) \cup \mathcal{R}) \subseteq \mathsf{Clo}^S_R(\mathcal{R}) \cup \{\, \psi \wedge \varphi \to \chi \mid \psi \wedge \varphi' \to \chi \in \mathsf{Clo}^S_{RF}(\mathcal{D}),\ \varphi \to \varphi' \in \mathsf{Clo}^S_R(\mathcal{R}) \,\}$ (by Lemma B.12). By Lemma B.16, $r \in \mathsf{Clo}^S_R(\mathcal{R}) \cup \{\, \psi \wedge \varphi \to \chi \mid \psi \wedge \varphi' \to \chi \in \mathsf{Clo}^S_F(\mathsf{Clo}_R(\mathcal{D})),\ \varphi \to \varphi' \in \mathsf{Clo}^S_R(\mathcal{R}) \,\} = \mathsf{Clo}^S_F(\mathsf{Clo}_R(\mathcal{R}) \cup \{\, \psi \wedge \varphi \to \chi \mid \psi \wedge \varphi' \to \chi \in \mathsf{Clo}_R(\mathcal{D}),\ \varphi \to \varphi' \in \mathsf{Clo}_R(\mathcal{R}) \,\})$. The claim, $r \in \mathsf{Clo}^S_F(\mathsf{Clo}_R(\mathcal{D} \cup \mathcal{R}))$, follows by Lemma B.11. □

Lemma B.18. *Let $\mathcal{D}$ be a separable $\mathcal{ELU}$-program. Let, for every $A \in \Sigma_c(\mathcal{D})$, Q_A be the predicate introduced for A in $\Omega(\mathcal{D})$. Let $\mathcal{D}' = \mathcal{D} \cup \bigcup_{A' \in \Sigma_c(\mathcal{D})} \{A'(x) \to Q_{A'}(x)\}$ and let $r = A \alpha Q_B$ be a rule where $A, B \in \Sigma_c(\mathcal{D})$. Then:*

$$r \in \mathsf{Clo}^S_{RF}(\mathcal{D}') \text{ iff } r \in \mathsf{Clo}^S_R(\mathsf{Clo}_{RF}(\mathcal{D}_\vee \cup \Omega(\mathcal{D}))_H \cup \Xi(\mathcal{D}))$$

Proof. Suppose $r \in \mathsf{Clo}_{RF}(\mathcal{D}')$. By Lemma B.16, there is some $r' \in \mathsf{Clo}_R(\mathcal{D}')$ such that $r \in \mathsf{Clo}_F^S(\{r'\})$. By Lemma B.6, either $r' \in \mathsf{Clo}_R(\mathcal{D}'_\forall)$ or $r' = r_\vee + r_1 + \ldots + r_n$ where $r_\vee \in \mathsf{Clo}_R(\mathcal{D}'_\vee)$ and $r = A_a\alpha_i Q_B \in \mathsf{Clo}_R(\mathcal{D}'_\forall)$ $(i \in [1,n])$. We show the claim for the second case as the first case is similar but simpler. By Lemma B.14, we have $r_i \in \mathsf{Clo}_R^S(\{A_i(y) \wedge N_{e_{\langle A_i,Q_B\rangle}}(x,y) \to B(x)\} \cup \mathcal{P}_{e_{\langle A_i,Q_B\rangle}}) \subseteq \mathsf{Clo}_R^S(\mathcal{D}_\vee \cup \Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$ $(i \in [1,n])$. Therefore, $r' \in \mathsf{Clo}_R^S(\mathcal{D}_\vee \cup \Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$ and, consequently, $r \in \mathsf{Clo}_{RF}^S(\mathcal{D}_\vee \cup \Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$. The claim follows by Lemma B.13.

Now suppose $r \in \mathsf{Clo}_R(\mathsf{Clo}_{RF}(\mathcal{D}_\vee \cup \Omega(\mathcal{D}))_H \cup \Xi(\mathcal{D}))$. By Lemma B.13, we then have $r \in \mathsf{Clo}_{RF}^S(\mathcal{D}_\vee \cup \Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$. By Lemma B.17, there is some $r' \in \mathsf{Clo}_R(\mathcal{D}_\vee \cup \Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$ such that $r \in \mathsf{Clo}_F^S(\{r'\})$. By Lemma B.6, either $r' \in \mathsf{Clo}_R(\Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$ or $r' = r_\vee + r_1 + \ldots + r_n$ such that $r_\vee \in \mathsf{Clo}_R(\mathcal{D}_\vee)$ and $\{r_1, \ldots, r_n\} \subseteq \mathsf{Clo}_R(\Omega(\mathcal{D}) \cup \Xi(\mathcal{D}))$.

Let $\mathcal{P}' = \bigcup_{A' \in \Sigma_c(\mathcal{D})} \{A'(x) \to Q_{A'}(x)\}$. It suffices to show that for every $i \in [1,n]$: $r_i \in \mathsf{Clo}_R^S(\mathcal{D}_\forall \cup \mathcal{P}')$ (since then $r \in \mathsf{Clo}_F^S(\mathsf{Clo}_R^S(\mathcal{D}_\vee \cup \mathcal{D}_\forall \cup \mathcal{P}')) = \mathsf{Clo}_{RF}^S(\mathcal{D}')$). So, let $i \in [1,n]$ and let $r_i = A_i\alpha_i Q_B$. By definition, no two rules in $\Omega(\mathcal{D})$ resolve with each other. Moreover, the rules in $\Omega(\mathcal{D})$ have pairwise distinct unary predicates in their bodies. Therefore, with Lemma B.11, we obtain $r_i \in \mathsf{Clo}_R^S(\{A_i(y) \wedge N_{e_{\langle A_i,Q_B\rangle}}(x,y) \to Q_B(x)\} \cup \Xi(\mathcal{D}))$. The claim follows by Lemma B.14 and Proposition B.10. □

Proposition B.19. *Let $\mathcal{F}$ be a set of FO-sentences and let r be a Datalog rule of the form $\varphi \to H$. Then $\mathcal{F} \models r$ if and only if for every substitution σ from the variables in r to individuals: $\mathcal{F} \cup \{\varphi\sigma\} \models H\sigma$.*

Lemma B.20. *Let $\mathcal{D}$ be an $\mathcal{ELU}$-program, let $\mathcal{P}$ be an $\mathcal{EL}$-program, and let Q be a unary predicate such that $\Sigma(\mathcal{D}) \subseteq \Sigma(\mathcal{P})$ and for every Datalog rule $r = \varphi \to Q(x)$ over $\Sigma(\mathcal{D})$:*

$$r \in \mathsf{Clo}_{RF}^S(\mathcal{D}) \text{ iff } r \in \mathsf{Clo}_R^S(\mathcal{P})$$

Then for every ABox $\mathcal{A}$ over $\Sigma(\mathcal{D})$ and every individual a:

$$\mathcal{D} \cup \mathcal{A} \models Q(a) \text{ iff } \mathcal{P} \cup \mathcal{A} \models Q(a)$$

Proof. Suppose $\mathcal{D} \cup \mathcal{A} \models Q(a)$. Let θ be a substitution mapping each individual in $\mathcal{A}$ to a distinct variable and let $r = (\bigwedge \mathcal{A})\theta \to Q(a\theta)$. By Proposition B.19, we then have $\mathcal{D} \models r$. Then, by completeness of binary resolution with positive factoring, $r \in \mathsf{Clo}_{RF}^S(\mathcal{D})$, and hence $r \in \mathsf{Clo}_R^S(\mathcal{P})$ by the assumption. Since binary resolution with factoring is a sound inference calculus, we have $\mathcal{P} \models r$, and hence $\mathcal{D} \cup \mathcal{A} \models Q(a)$ by Proposition B.19 (as the inverse of θ is a substitution from variables in r to individuals).

The other direction follows analogously since binary resolution without factoring is a sound and complete inference calculus for Horn clauses. □

Lemma 4.13. *Let $\mathcal{D}$ be a separable $\mathcal{ELU}$-program. Let $\mathcal{P}$ be the union of $\Xi(\mathcal{D})$ and the output of Procedure 1 on $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$. Then, for every query $A(x)$ and every ABox $\mathcal{A}$, we have $\mathsf{cert}(A(x), \mathcal{D}, \mathcal{A}) = \mathsf{cert}(Q_A(x), \mathcal{P}, \mathcal{A})$.*

Proof. Let $\mathcal{D}' = \mathcal{D} \cup \bigcup_{A \in \Sigma_c(\mathcal{D})} \{A(x) \to Q_A(x)\}$ where Q_A is the predicate introduced in $\Omega(\mathcal{D})$ for A. Clearly, we have $\mathcal{D} \cup \mathcal{A} \models A(a)$ iff $\mathcal{D}' \cup \mathcal{A} \models Q_A(a)$ for every query $A(x)$, individual a, and ABox $\mathcal{A}$ over $\Sigma(\mathcal{D})$. Hence, it suffices to show that for every $A(x)$, a, $\mathcal{A}$: $\mathcal{D}' \cup \mathcal{A} \models Q_A(a)$ iff $\mathcal{P} \cup \mathcal{A} \models Q_A(a)$. By Lemma B.20, this is the case if $\mathsf{Clo}^S_{RF}(\mathcal{D}') = \mathsf{Clo}^S_R(\mathcal{P})$ restricted to Datalog rules over $\Sigma(\mathcal{D}')$ of the form $\varphi \to Q_A(x)$ (for every $A(x)$). The equation holds by Lemma B.18 as every Datalog rule over $\Sigma(\mathcal{D}')$ in $\mathsf{Clo}^S_{RF}(\mathcal{D}') \cup \mathsf{Clo}^S_R(\mathcal{P})$ is linear, and $\mathsf{Clo}_{RF}(\mathcal{D}_\vee \cup \Omega(\mathcal{D}))_H$ is equal to the output of Procedure 1 on $\mathcal{D}_\vee \cup \Omega(\mathcal{D})$ up to subsumption. □

Theorem 4.14. *On input a separable $\mathcal{ELU}$-program $\mathcal{D}$, Procedure 3 returns a Datalog rewriting of $\mathcal{D}$.*

Proof. Let $\mathcal{P}$ be set computed in Line (9) of Procedure 3 on input $\mathcal{D}$ and let $\mathcal{P}'$ be the output of the procedure. By Lemma 4.13, it suffices to show that $\mathsf{cert}(Q_A(x), \mathcal{P}, \mathcal{A}) = \mathsf{cert}(A(x), \mathcal{P}', \mathcal{A})$ for every ABox $\mathcal{A}$ and every unary predicate A in $\mathcal{D}$. So, let $\mathcal{A}$ and A be as required.

Clearly, $\mathsf{cert}(Q_A(x), \mathcal{P}, \mathcal{A}) \subseteq \mathsf{cert}(A(x), \mathcal{P}', \mathcal{A})$. For the other inclusion, observe that, by Lemma 4.13, $\mathcal{D} \cup \mathcal{A} \models A(a)$ whenever $\mathcal{P} \cup \mathcal{A} \models Q_A(a)$. So, replacing Q_A by A in $\mathcal{P}$ does not allow to derive anything that is not entailed by the original program. □