Datalog Rewritability of Disjunctive Datalog Programs and Non-Horn Ontologies[☆]

Mark Kaminski, Yavor Nenov, Bernardo Cuenca Grau

Department of Computer Science, University of Oxford, UK

Abstract

We study the problem of rewriting a Disjunctive Datalog program into an equivalent plain Datalog program (i.e., one that entails the same facts for every dataset). We show that a Disjunctive Datalog program is Datalog rewritable if and only if it can be rewritten into a linear program (i.e., having at most one IDB body atom in each rule), thus providing a novel characterisation of Datalog rewritability in terms of linearisability. Motivated by this result, we propose the class of markable programs, which extends both Datalog and linear Disjunctive Datalog and admits Datalog rewritings of polynomial size. We show that our results can be seamlessly applied to ontological reasoning and identify two classes of non-Horn ontologies that admit Datalog rewritings of polynomial and exponential size, respectively. Finally, we shift our attention to conjunctive query answering and extend our results to the problem of computing a rewriting of a Disjunctive Datalog program that yields the same answers to a given query w.r.t. arbitrary data. Our empirical results suggest that a fair number of non-Horn ontologies are Datalog rewritable and that query answering over such ontologies becomes feasible using a Datalog engine.

Keywords: Knowledge Representation and Reasoning, Rule Languages, Disjunctive Datalog, Query Answering, Datalog Rewritability, Description Logics, Ontologies.

1. Introduction

Disjunctive Datalog, which extends plain Datalog by allowing disjunction in the head of rules, is a powerful knowledge representation formalism that has found many applications in the areas of deductive databases, information integration, and ontological reasoning [3, 4]. Expressiveness comes, however, at the expense of computational cost: fact entailment is co-NExpTime-complete in

[☆]This is an extended version of two papers presented at AAAI 2014 [1] and RR 2014 [2].

Email addresses: `mark.kaminski@cs.ox.ac.uk` (Mark Kaminski), `yavor.nenov@cs.ox.ac.uk` (Yavor Nenov), `bernardo.cuenca.grau@cs.ox.ac.uk` (Bernardo Cuenca Grau)

combined complexity and co-NP-complete with respect to data [3]. Thus, even with the development of optimised implementations [5, 6], robust behaviour of reasoners in data-intensive applications cannot be guaranteed.

Plain Datalog offers more favourable computational properties at the expense of a loss in expressive power, namely ExpTime-completeness in combined complexity and PTime-completeness in data [4]. Tractability in data complexity is an appealing property for data-intensive applications of ontologies. In particular, the RL profile of the ontology language OWL 2 was designed so that each ontology corresponds to a Datalog program [7]. Furthermore, Datalog programs obtained from RL ontologies contain rules of a restricted shape, and hence can be evaluated in polynomial time also in combined complexity, thus providing the ground for robust implementations. The standardisation of OWL 2 RL has spurred the development of reasoning engines within industry and academia, such as GraphDB [8] (formerly known as OWLIM), Oracle's RDF Semantic Graph [9], and RDFox [10].

In this paper, we study the problem of rewriting a Disjunctive Datalog program into a Datalog program that entails the same facts for every dataset. By computing such rewritings, not only can we ensure tractability in data, but also exploit the highly optimised reasoning infrastructure available for Datalog.

Our first contribution is a novel characterisation of Datalog rewritability based on *linearity*: a restriction that requires each rule to contain at most one body atom with an IDB predicate (i.e., a predicate occurring in head position). For plain Datalog, linearity is known to limit the effect of recursion and lead to reduced data and combined complexity of reasoning [11]. For Disjunctive Datalog programs the effects of the linearity restriction are, to the best of our knowledge, unknown.

In Section 3, we show that every linear Disjunctive Datalog program admits a Datalog rewriting of polynomial size; conversely, every Datalog program can be polynomially rewritten into linear Disjunctive Datalog. We thus show that linear Disjunctive Datalog and Datalog have the same computational properties, and linearisability of Disjunctive Datalog programs is equivalent to their rewritability into Datalog. We establish these results by means of *program transposition*—a novel polynomial transformation in which the rules of a given Disjunctive Datalog program $\mathcal{P}$ are inverted by moving all IDB atoms between head and body while at the same time replacing their corresponding predicates with relevant auxiliary predicates of higher arity. If $\mathcal{P}$ is linear, transposition yields a Datalog program; conversely, if $\mathcal{P}$ is Datalog, then transposition yields a linear program.

Motivated by these results we propose in Section 4 the class of *markable* Disjunctive Datalog programs, which extends both Datalog and linear Disjunctive Datalog while at the same time ensuring that polynomial Datalog rewritings can be computed by a refinement of transposition. The idea behind markable programs stems from a natural relaxation of the linearity requirement: instead of applying to all IDB predicates, it applies only to a subset of *marked* IDB predicates. We show that our extended class of programs is efficiently recognisable via a reduction to 2-SAT and that each markable program admits a polynomial Datalog rewriting. In this way, our language based on markability is capable of capturing disjunctive information while retaining the favourable computational properties of Datalog.

Unfortunately, in the case of programs that do not satisfy our markability condition we have no algorithmic means to determine whether they can be rewritten into Datalog; indeed, checking Datalog rewritability (or equivalently, linearisability) of Disjunctive Datalog programs is undecidable. To go a step farther and identify an even larger class of rewritable programs, we propose in Section 5 a linearisation procedure based on program unfolding transformations [12, 13]. Our procedure picks a non-markable rule and a "culprit" body atom and replaces it with markable rules by unfolding the selected atom. Our procedure is sound but incomplete: if it succeeds, it outputs a markable program that is then rewritten into Datalog; if it fails, no conclusion can be drawn.

In Section 6 we study the applicability of our results to ontology reasoning. We first consider the natural syntactic intersection between OWL 2 and Disjunctive Datalog (which we call $\mathrm{RL}^{\sqcup}$), and show that fact entailment over $\mathrm{RL}^{\sqcup}$ ontologies corresponding to a markable program is tractable in combined complexity (and hence no harder than in OWL 2 RL [7]). We then lift the markability condition to ontologies with existentially quantified axioms, and show that markable ontologies in the expressive Description Logic $\mathcal{SHI}$ admit an exponential size Datalog rewriting.

In Section 7, we shift our attention to conjunctive query answering. In this setting, it is no longer possible to obtain query-independent Datalog rewritings. Lutz and Wolter [14] showed that for *any* program containing at least one disjunctive rule there exists a conjunctive query such that answering the (fixed) query with respect to the (fixed) program and an input dataset is co-NP-hard; thus, under standard complexity-theoretic assumptions, no Datalog rewriting for such query and program exists. We therefore propose classes of conjunctive queries and Disjunctive Datalog programs that admit Datalog rewritings and discuss the implications of these results for ontology reasoning.

We have implemented and evaluated our techniques on a wide range of ontologies. Our experiments indicate that a fair number of non-Horn ontologies used in practice admit a Datalog rewriting. Additionally, our experiments also demonstrate that our rewriting techniques can significantly improve reasoning performance and robustness in practice.

2. Preliminaries

We assume standard first-order logic notions of terms, atoms, literals, formulae, sentences, and entailment. We also consider basic notions in first-order theorem proving such as substitution, unification, most general unifiers (MGUs), clauses, and clause subsumption [15]. *Positive factoring* (PF) and *binary resolution* (BR) are the following inference rules, where σ is an MGU of atoms A and B:

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

Rule languages such as Disjunctive Datalog typically support non-monotonic negation as failure in the Logic Programming literature [4]. In this paper, however, our focus is on monotonic reasoning and hence we will restrict ourselves to rules in the context of first-order logic, where negation as failure is not allowed.

2.1. Rule Languages

We assume a first-order signature Σ (with function symbols) where the set of predicates is partitioned into *extensional* (or *EDB*) and *intensional* (or *IDB*). We say that an atom is EDB (IDB) if so is its predicate.

A *fact* is a function-free ground EDB atom over Σ and a *dataset* is a finite set of facts. A *rule* (or first-order clause) is a sentence over Σ of the form

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

where tuples of variables $\vec{x}$ and $\vec{z}$ are disjoint, $\varphi(\vec{x},\vec{z})$ is a (possibly empty) conjunction of distinct atoms over variables $\vec{x} \cup \vec{z}$, and $\psi(\vec{x})$ is a (possibly empty) disjunction of distinct IDB atoms over $\vec{x}$. Formula φ is the *body* of r, and ψ is the *head*. Quantifiers in rules are omitted for brevity. We assume that rules are *safe*, i.e., all variables in the head of a rule occur in the body.

We say that a rule is

- *Horn* if its head consists of at most one atom, and *disjunctive* otherwise;
- *Disjunctive Datalog* if it is function-free;
- *Datalog* if it is both Disjunctive Datalog and Horn; and
- *linear* if it contains at most one IDB atom in the body.

A *program* is a finite set of rules. Additionally, we say that a program is of one of the aforementioned types if so are all of its rules.

We conclude this section with a few remarks on the structure of programs that will allow us to substantially simplify later on the presentation of our technical results. These considerations are without loss of generality.

1. The reader may have noticed that the restriction that IDB predicates do not occur in datasets is not typically adopted in AI applications. This assumption can be seamlessly lifted as explained next (see, e.g., [16]). Let θ be a predicate substitution mapping each IDB predicate Q in $\mathcal{P}$ to a fresh predicate Q'. The *IDB expansion* $\mathcal{P}^e$ of $\mathcal{P}$ is obtained by first applying θ to $\mathcal{P}$ and then adding a rule $Q(\vec{x}) \rightarrow Q'(\vec{x})$ for each IDB predicate Q, where $\vec{x}$ is a vector of distinct variables and where Q is now treated as an EDB predicate. Then, for each dataset $\mathcal{D}$ and each formula φ over the signature of $\mathcal{P}$, we have $\mathcal{P} \cup \mathcal{D} \models \varphi$ iff $\mathcal{P}^e \cup \mathcal{D} \models \varphi\theta$. Thus, we can lift all our results in this paper to allow for IDB atoms in datasets by simply replacing $\mathcal{P}$ with its IDB expansion.
2. We use the language of first-order rules to capture prominent Knowledge Representation formalisms such as Description Logics. To this end, we require that signatures Σ contain the special predicates in first-order logic for

universal truth $\top$, and falsehood $\bot$. For convenience, however, we treat them in a non-standard way as ordinary predicates and assume that their special meaning is axiomatised. Specifically, we assume that $\top$ is unary and EDB, whereas $\bot$ is nullary and IDB. We will assume that every dataset $\mathcal{D}$ and program $\mathcal{P}$ contain a fact $\top(a)$ for each constant in their signature. This assumption allows us to treat $\top$ as a proper EDB predicate. Furthermore, every program $\mathcal{P}$ is of the form $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}^{\bot}$, where each rule in $\mathcal{P}_0$ has a non-empty head and does not mention $\bot$ in the body, and where $\mathcal{P}^{\bot}$ is empty if $\bot$ does not occur in $\mathcal{P}_0$ and it consists of a single rule with $\bot$ in the body and an empty head otherwise.

2.2. Hyperresolution

Reasoning w.r.t. programs can be realised by means of the *hyperresolution calculus* (also referred to as *forward chaining* in the literature) [15–17]. Let $r = \bigwedge_{i=1}^{n} \beta_i \to \varphi$ be a rule and, for each $1 \leq i \leq n$, let ψ_i be a disjunction of ground atoms $\psi_i = \chi_i \vee \alpha_i$ with α_i a single atom. Let σ be an MGU of each β_i, α_i. Then, the disjunction of ground atoms $\varphi' = \varphi\sigma \vee \chi_1 \vee \cdots \vee \chi_n$ is a *hyperresolvent* of r and $\psi_1, \ldots, \psi_n$.[1]

Let $\mathcal{P}$ be a program, let $\mathcal{D}$ be a dataset, and let φ be a disjunction of ground atoms. A (hyperresolution) *derivation* of φ from $\mathcal{P} \cup \mathcal{D}$ is a pair $\rho = (T, \lambda)$, where T is a tree, λ a labeling function mapping each node in T to a disjunction of ground atoms, and the following properties hold for each $v \in T$:

1. if v is the root of T then $\lambda(v) = \varphi$;
2. if v is a leaf in T then either $\lambda(v) \in \mathcal{D}$ or $(\to \lambda(v)) \in \mathcal{P}$; and
3. if v has children $w_1, \ldots, w_n$, then $\lambda(v)$ is a hyperresolvent of a rule $r \in \mathcal{P}$ and $\lambda(w_1), \ldots, \lambda(w_n)$.

We call a node in a derivation EDB (resp. IDB) if it is labeled by a single EDB fact (resp. a disjunction of IDB atoms); it follows from the definition of a derivation that every node is either EDB or IDB.

We write $\mathcal{P} \cup \mathcal{D} \vdash \varphi$ to denote that φ has a derivation from $\mathcal{P} \cup \mathcal{D}$. Then, hyperresolution is sound and complete in the following sense: $\mathcal{P} \cup \mathcal{D}$ is unsatisfiable iff $\mathcal{P} \cup \mathcal{D} \vdash \bot$; furthermore, if $\mathcal{P} \cup \mathcal{D}$ is satisfiable then $\mathcal{P} \cup \mathcal{D} \models \alpha$ iff $\mathcal{P} \cup \mathcal{D} \vdash \alpha$ for every ground atom α.

2.3. Queries, Reasoning Problems, and Rewritings

A *conjunctive query* (*CQ*) q is a formula of the form $\exists \vec{y}.\varphi(\vec{x}, \vec{y})$, with φ a conjunction of function-free atoms. A CQ is *Boolean* if it is a sentence, and it is *atomic* if $\varphi(\vec{x}, \vec{y})$ consists of a single atom and $\vec{y}$ is empty.

[1]We treat disjunctions of atoms as sets and hence we do not allow for duplicated atoms in a disjunction.

Let $\mathcal{L}$ be a class of programs. *Fact entailment* w.r.t. $\mathcal{L}$ is the problem of deciding whether $\mathcal{P} \cup \mathcal{D} \models \alpha$ when given as input a program $\mathcal{P} \in \mathcal{L}$, a dataset $\mathcal{D}$ and a fact α. The set of all facts entailed by $\mathcal{P} \cup \mathcal{D}$ is typically referred to as the *materialisation* of $\mathcal{P} \cup \mathcal{D}$. Similarly, *CQ entailment* w.r.t. $\mathcal{L}$ is the problem of deciding whether $\mathcal{P} \cup \mathcal{D} \models q$ when given as input $\mathcal{P} \in \mathcal{L}$, a dataset $\mathcal{D}$ and a Boolean CQ q. CQ entailment reduces to fact entailment if we require the input query q to be atomic.

The computational properties of these problems are well-understood (e.g., see [4]). Both problems are undecidable already for the the class of Horn programs (which may contain function symbols). For Disjunctive Datalog, they are co-NExpTime-complete in combined complexity and co-NP-complete w.r.t. data. Finally, for plain Datalog they are ExpTime-complete in combined complexity and PTime-complete w.r.t. data.

Finally, we say that a program $\mathcal{P}$ is a *rewriting* of a CQ q w.r.t. a set of first-order sentences $\mathcal{F}$ if there exists a predicate A_q such that for each dataset $\mathcal{D}$ over the signature of $\mathcal{F}$ and each tuple of constants $\vec{a}$ we have $\mathcal{F} \cup \mathcal{D} \models q(\vec{a})$ iff $\mathcal{P} \cup \mathcal{D} \models A_q(\vec{a})$. Program $\mathcal{P}$ is a rewriting of $\mathcal{F}$ if it is a rewriting of every atomic query over the signature of $\mathcal{F}$. In particular, if $\mathcal{F}$ is a program, then $\mathcal{P} \cup \mathcal{D}$ and $\mathcal{F} \cup \mathcal{D}$ are equisatisfiable for each dataset $\mathcal{D}$.

3. A Characterisation of Datalog Rewritability

In this section we establish a strong correspondence between linear Disjunctive Datalog and plain Datalog. This correspondence leads to a new characterisation of Datalog rewritability: a Disjunctive Datalog program is Datalog rewritable if and only if it is rewritable into linear Disjunctive Datalog.

Our results stem from the correctness of *program transposition*: a novel polynomial transformation Ξ applicable to an arbitrary Disjunctive Datalog program $\mathcal{P}$ in which the rules of $\mathcal{P}$ are inverted by moving all IDB atoms from head to body and vice versa while at the same time replacing their corresponding predicates by auxiliary predicates of higher arity. Intuitively, each fact over an auxiliary predicate captures a relevant dependency in hyperresolution proofs between the corresponding facts over predicates in $\mathcal{P}$. Such dependencies are then "propagated" by the transposed rules so that the derivation of each fact in $\mathcal{P} \cup \mathcal{D}$ can be captured by a derivation in $\Xi(\mathcal{P}) \cup \mathcal{D}$, with $\Xi(\mathcal{P})$ the transposition of $\mathcal{P}$. In this way, transposition preserves fact entailment: for every dataset $\mathcal{D}$ and fact α over the predicates of the original program $\mathcal{P}$, it holds that $\mathcal{P} \cup \mathcal{D}$ entails α if and only if so does $\Xi(\mathcal{P}) \cup \mathcal{D}$. Program transposition is presented in Section 3.1 and its correctness is established in Section 3.2.

In Section 3.3, we exploit transposition to establish our characterisation of Datalog rewritability of Disjunctive Datalog programs. It follows from the definition of transposition that $\Xi(\mathcal{P})$ is Datalog whenever $\mathcal{P}$ is a linear Disjunctive Datalog program; conversely, $\Xi(\mathcal{P})$ is linear whenever $\mathcal{P}$ is Datalog. Consequently, Ξ can be directly exploited to polynomially rewrite linear Disjunctive Datalog programs into Datalog, and vice versa. In this way, we not only can conclude that fact entailment over linear Disjunctive Datalog programs has exactly the same data and combined complexity as over plain Datalog programs, but also that a Disjunctive Datalog program admits a Datalog rewriting if and only if it is linearisable. Datalog rewritability and linearisability of Disjunctive Datalog programs are thus equivalent problems.

3.1. Program Transposition

To motivate program transposition, consider as an example the following Disjunctive Datalog program $\mathcal{P}_1$, which we aim to rewrite into plain Datalog:

$$\begin{aligned} \mathcal{P}_1 = \{\, & C(x) \to B(x) \vee G(x), && (1)\\ & G(y) \wedge E(x,y) \to B(x), && (2)\\ & B(y) \wedge E(x,y) \to G(x) \,\} && (3) \end{aligned}$$

Intuitively, this program encodes non-2-colourability: a property of graphs which is expressible in plain Datalog [18]. Every dataset $\mathcal{D}$ encoding a graph with (symmetric) edge relation E and vertex predicate C is non-2-colourable if and only if $\mathcal{P}_1 \cup \mathcal{D}$ entails any fact over the IDB predicates B and G.

We next construct the transposition $\Xi(\mathcal{P}_1)$ of $\mathcal{P}_1$. The transposed program $\Xi(\mathcal{P}_1)$ will be a rewriting of $\mathcal{P}_1$: for any dataset $\mathcal{D}$ over the EDB predicates E and C and fact α over the signature of $\mathcal{P}_1$, we have $\mathcal{P}_1 \cup \mathcal{D} \models \alpha$ iff $\Xi(\mathcal{P}_1) \cup \mathcal{D} \models \alpha$.

Since predicates C and E are EDB, their extension w.r.t. $\mathcal{D}$ depends solely on the facts in $\mathcal{D}$; as a result, the EDB atoms in $\mathcal{P}$ will remain unaffected by the transposition. To ensure that $\mathcal{P}_1$ and $\Xi(\mathcal{P}_1)$ entail the same IDB facts we introduce in $\Xi(\mathcal{P}_1)$ fresh binary predicates $\overline{B}^G$, $\overline{B}^B$, $\overline{G}^B$, and $\overline{G}^G$. The intended meaning of these auxiliary predicates is as follows: if a fact $\overline{X}^Y(c,d)$ holds in $\Xi(\mathcal{P}_1) \cup \mathcal{D}$, with $X, Y \in \{B, G\}$, then proving $X(c)$ suffices for proving $Y(d)$ in $\mathcal{P}_1 \cup \mathcal{D}$; that is, $\mathcal{P}_1 \cup \mathcal{D}$ logically entails the propositional implication $X(c) \to Y(d)$. The key step in our transformation is then to flip the direction of all rules in $\mathcal{P}_1$ by moving all IDB atoms from the head to the body and vice versa while at the same time replacing their predicates with the relevant auxiliary predicates of higher arity. Since $\mathcal{P}_1$ is linear, our transformation will ensure that the resulting rules are Datalog and hence have a single atom in the head. In particular, Rule (2) leads to the following two rules in $\Xi(\mathcal{P}_1)$:

$$\overline{B}^X(x,z) \wedge E(x,y) \to \overline{G}^X(y,z) \quad \text{for each } X \in \{G, B\} \tag{2'}$$

These rules are natural consequences of Rule (2) under the intended meaning of the auxiliary predicates: if we can prove a goal $X(z)$ by proving first $B(x)$ and $E(x,y)$, then by Rule (2) we deduce that proving $G(y)$ suffices to prove $X(z)$.

In contrast to (2), Rule (1) contains no IDB body atoms. We transpose this rule in a slightly different way by introducing the following rules:

$$C(x) \wedge \overline{B}^X(x,z) \wedge \overline{G}^X(x,z) \to X(z) \quad \text{for each } X \in \{G, B\} \tag{1'}$$

Similarly to the previous case, this rule follows from Rule (1): if $C(x)$ holds and we can establish that $X(z)$ can be proved from $B(x)$ and also from $G(x)$, then $X(z)$ must hold.

Finally, since $\mathcal{D}$ does not contain any facts over the auxiliary predicates, Rules (2') and (1') are not applicable. Therefore, we introduce the following rules in $\Xi(\mathcal{P}_1)$ in order to "initialise" the extension of these auxiliary predicates:

$$\top(x) \rightarrow \overline{X}^X(x,x) \quad \text{for each } X \in \{G, B\} \tag{4}$$

Rules (4) encode tautological information under the intended meaning of the auxiliary predicates (a fact $\overline{B}^B(c,c)$ intuitively means that $B(c)$ suffices to prove itself). The initialisation rules ensure that all auxiliary predicates are instantiated and the remaining rules in the transposed program become applicable.

In sum, $\Xi(\mathcal{P}_1)$ is the following Datalog program, where each rule mentioning X stands for one rule where $X = B$ and one where $X = G$:

$$\begin{aligned}
\Xi(\mathcal{P}_1) = \{\, & C(x) \wedge \overline{B}^X(x,z) \wedge \overline{G}^X(x,z) \rightarrow X(z), && (1')\\
& \overline{B}^X(x,z) \wedge E(x,y) \rightarrow \overline{G}^X(y,z), && (2')\\
& \overline{G}^X(x,z) \wedge E(x,y) \rightarrow \overline{B}^X(y,z), && (3')\\
& \top(x) \rightarrow \overline{X}^X(x,x) \,\} && (4)
\end{aligned}$$

Let us now consider the following Datalog program $\mathcal{P}_2$, which encodes path system accessibility (a canonical PTime-complete problem [19]):

$$\begin{aligned}
\mathcal{P}_2 = \{\, & V(x) \rightarrow A(x), && (5)\\
& R(x,y,z) \wedge A(y) \wedge A(z) \rightarrow A(x) \,\} && (6)
\end{aligned}$$

Note that $\mathcal{P}_2$ is not linear since Rule (6) contains two IDB atoms in the body. Furthermore, reasoning in linear Datalog is NLogSpace-complete in data and hence, under standard complexity-theoretic assumptions, $\mathcal{P}_2$ cannot be linearised within plain Datalog. In contrast, we can exploit transposition to rewrite $\mathcal{P}_2$ into the following linear Disjunctive Datalog program $\Xi(\mathcal{P}_2)$:

$$\begin{aligned}
\Xi(\mathcal{P}_2) = \{\, & V(x) \wedge \overline{A}^A(x,y) \rightarrow A(y), && (5')\\
& R(x,y,z) \wedge \overline{A}^A(x,u) \rightarrow \overline{A}^A(y,u) \vee \overline{A}^A(z,u), && (6')\\
& \top(x) \rightarrow \overline{A}^A(x,x) \,\} && (7)
\end{aligned}$$

As in the previous example, the predicate $\overline{A}^A$ is fresh and carries the same intended meaning as before: if $\Xi(\mathcal{P}_2) \cup \mathcal{D}$ entails a disjunction $\bigvee_i \overline{A}^A(c_i, d)$, then $\mathcal{P}_2 \cup \mathcal{D}$ entails the implication $(\bigwedge_i A(c_i)) \rightarrow A(d)$. The initialisation rule (7) is again used to instantiate the auxiliary predicate, whereas Rules (5') and (6') are obtained by transposing (5) and (6), respectively. Rules (5') and (6') in $\Xi(\mathcal{P}_2)$ are natural consequences of their counterparts in $\mathcal{P}_2$ under the intended meaning of the fresh predicate. For example, to justify Rule (6'), assume that $R(a,b,c)$ holds and $A(d)$ follows from $A(a)$; then, by Rule (6) we can conclude that $A(d)$ also follows from $A(b) \wedge A(c)$.

We are now ready to define program transposition formally. Transposition is a quadratic transformation and the arity of predicates is at most doubled.

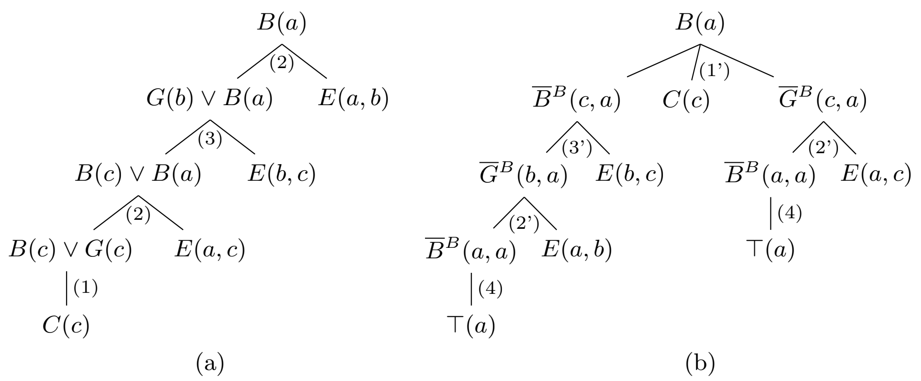


Figure 1: (a) derivation of $B(a)$ from $\mathcal{P}_1 \cup \mathcal{D}_1$; (b) derivation of $B(a)$ from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$

Definition 1. Let $\mathcal{P}$ be a Disjunctive Datalog program. For each pair (P, Q) of IDB predicates in $\mathcal{P}$, let $\overline{P}^Q$ be a fresh predicate of arity $\mathsf{arity}(P) + \mathsf{arity}(Q)$. The transposition of $\mathcal{P}$ is the smallest program $\Xi(\mathcal{P})$ containing all Rules 1–3 given next, where $\xi_\top$ is the least conjunction of $\top$-atoms needed to make a rule safe, φ is the conjunction of all EDB atoms in a rule, all predicates P_i are IDB, and $\vec{y} = y_1 \dots y_{\mathsf{arity}(R)}$ is a vector of distinct fresh variables:

1. $\xi_\top \to \overline{R}^R(\vec{y}, \vec{y})$ for each IDB predicate R;
2. $\xi_\top \wedge \varphi \wedge \bigwedge_i \overline{P}_i^R(\vec{s}_i, \vec{y}) \to \bigvee_j \overline{Q}_j^R(\vec{t}_j, \vec{y})$ for each IDB predicate R and rule in $\mathcal{P}$ of the form $\varphi \wedge \bigwedge_j Q_j(\vec{t}_j) \to \bigvee_i P_i(\vec{s}_i)$ for $\bigwedge_j Q_j(\vec{t}_j)$ nonempty; and
3. $\varphi \wedge \bigwedge_i \overline{P}_i^R(\vec{s}_i, \vec{y}) \to R(\vec{y})$ for each IDB predicate R and each rule in $\mathcal{P}$ of the form $\varphi \to \bigvee_i P_i(\vec{s}_i)$. ◇

It follows from Definition 1 that program transposition can be used to transform linear Disjunctive Datalog programs into plain Datalog, and vice versa.

Proposition 2. *Let $\mathcal{P}$ be a Disjunctive Datalog program. Then:*

1. $\Xi(\mathcal{P})$ is Datalog if and only if $\mathcal{P}$ is linear;

2. $\Xi(\mathcal{P})$ is linear if and only if $\mathcal{P}$ is Datalog.

PROOF. Given a rule s, we write $|s|_\mathsf{h}$ for the number of atoms in the head of s and $|s|_\mathsf{b}$ for the number of IDB atoms in the body of s. Let $r \in \mathcal{P}$ and let r' be the transposition of r in $\Xi(\mathcal{P})$. Both claims follow since *(i)* $|r'|_\mathsf{h} = \max\{1, |r|_\mathsf{b}\}$, *(ii)* $|r'|_\mathsf{b} = |r|_\mathsf{h}$, and *(iii)* all rules in $\Xi(\mathcal{P})$ that are not derived from rules in $\mathcal{P}$ are both Datalog and linear. □

3.2. Correctness of Transposition

In this section we show the key property of transposition, namely that it preserves fact entailment for every dataset. Let us sketch the main intuitions on our example program $\mathcal{P}_1$ consisting of Rules (1)–(3) and $\mathcal{D}_1 = \{C(a), C(b), C(c), E(a,b), E(b,a)\ E(b,c), E(c,b)\ E(a,c), E(c,a)\}$.[2] Figure 1(a) shows a derivation ρ_1 of $B(a)$ from $\mathcal{P}_1 \cup \mathcal{D}_1$ while 1(b) shows a derivation ρ_2 of the same fact from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$. Recall that we represent derivations as trees whose nodes are labeled with disjunctions of facts and where each inner node is derived from its children using a rule of the program. In particular, to derive $B(a)$ in ρ_1 we start from fact $C(c)$ and use Rule (1) to obtain $B(c) \vee G(c)$; since fact $E(a,c)$ holds in $\mathcal{D}_1$, we can apply hyperresolution to the previously derived disjunction $B(c) \vee G(c)$, fact $E(a,c)$ and Rule (2) to derive $B(c) \vee B(a)$. Similarly, we can then obtain the disjunction $G(b) \vee B(a)$ using the fact $E(b,c)$ and Rule (3). Finally, Rule (2) allows us to resolve away fact $E(a,b)$ from $\mathcal{D}_1$ with the disjunct $G(b)$ from $G(b) \vee B(a)$ and derive $B(a) \vee B(a)$, which simplifies to $B(a)$.

We first show that if $B(a)$ is provable in $\mathcal{P}_1 \cup \mathcal{D}_1$ via a derivation such as ρ_1, then it is entailed by $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$. The crux of the proof is to show that each (disjunction of) IDB fact(s) in ρ_1 corresponds to (disjunctions of) facts over the auxiliary predicates entailed by $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$. These facts about the auxiliary predicates must be of the form $\overline{X}^B(u,a)$, where $B(a)$ is the goal, u is a constant, and $X \in \{B, G\}$. For example, $B(c) \vee G(c)$ in ρ_1 corresponds to facts $\overline{B}^B(c,a)$ and $\overline{G}^B(c,a)$, which are provable from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$, as witnessed by ρ_2.

Finally, we show the converse: if $B(a)$ is provable from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$ by a derivation such as ρ_2 then it follows from $\mathcal{P}_1 \cup \mathcal{D}_1$. For this, we show that each fact in ρ_2 about an auxiliary predicate carries the intended meaning, e.g., for $\overline{G}^B(b,a)$ we must have $\mathcal{P}_1 \cup \mathcal{D}_1 \models G(b) \to B(a)$.

Theorem 3. *Let $\mathcal{P}$ be a Disjunctive Datalog program. Then, program $\Xi(\mathcal{P})$ is a rewriting of $\mathcal{P}$ of polynomial size.*

Proof. We show the claim in two steps, which we outline next. For the rest of the proof, we fix an arbitrary dataset $\mathcal{D}$ over the EDB predicates of $\mathcal{P}$ and an IDB fact $P(\vec{a})$ (if P is EDB, then the claim is immediate).

1. In the first step, we show that $\mathcal{P} \cup \mathcal{D} \vdash P(\vec{a})$ implies $\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$. To this end, we consider an arbitrary derivation ρ of $P(\vec{a})$ from $\mathcal{P} \cup \mathcal{D}$ and show that $\Xi(\mathcal{P}) \cup \mathcal{D}$ entails $P(\vec{a}) \vee \bigwedge\{\, \bigvee_{i=1}^{n} \overline{Q}_i^P(\vec{b}_i, \vec{a}) \mid Q_i(\vec{b}_i) \in \lambda(v_i)\,\}$, where $\{v_1, \ldots, v_n\}$ is the set of all IDB leaves in ρ. We use this intermediate result to establish an inductive argument that proves $\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$.

2. In the second step, we show that $\Xi(\mathcal{P}) \cup \mathcal{D} \vdash P(\vec{a})$ implies $\mathcal{P} \cup \mathcal{D} \models P(\vec{a})$. We first observe that, given the form of the rules in $\Xi(\mathcal{P})$, each derivation of $P(\vec{a})$ from $\Xi(\mathcal{P}) \cup \mathcal{D}$ is *focused on* $P(\vec{a})$; that is, it contains only IDB atoms of the form $P(\vec{a})$ and $\overline{Q}^P(\vec{b}, \vec{a})$, for some Q and $\vec{b}$. Thus, every node v in a derivation of $P(\vec{a})$ from $\Xi(\mathcal{P}) \cup \mathcal{D}$ is labeled by a disjunction $\varphi \vee \bigvee_i \overline{Q}_i^P(\vec{b}_i, \vec{a})$, where φ implies $P(\vec{a})$. The claim then follows from the observation that $\mathcal{P} \cup \mathcal{D} \models \bigwedge_i Q(\vec{b}_i) \to P(\vec{a})$ for every such disjunction.

[2] In this and the following examples, we omit facts about $\top$ when enumerating the contents of a dataset.

The statement of the theorem follows directly from the properties shown in Steps 1 and 2 and the completeness of hyperresolution. We next show each step.

Step 1. Suppose $\mathcal{P} \cup \mathcal{D} \vdash P(\vec{a})$ and let $\rho = (T, \lambda)$ be a derivation of $P(\vec{a})$ from $\mathcal{P} \cup \mathcal{D}$. We show $\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$.

We call a nonempty subtree U of T an *upper portion* of T if *(i)* T and U have the same root and *(ii)* for each node in U, U contains all or none of its children in T. We begin by showing the following claim, where $\Psi(v_1, \ldots, v_n)$ denotes the set $\{ \bigvee_{i=1}^{n} \overline{Q}_i^P(\vec{b}_i, \vec{a}) \mid Q_i(\vec{b}_i) \in \lambda(v_i) \}$.

Claim ($\diamondsuit$). Let U be an upper portion of T and let $\{v_1, \ldots, v_n\}$ be the IDB leaves of U. Then

$$\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a}) \vee \bigwedge \Psi(v_1, \ldots, v_n)$$

We show the claim inductively. For the base case, suppose U consists of a single node. Then the claim reduces to $\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a}) \vee \overline{P}^P(\vec{a}, \vec{a})$, which follows since $\xi_\top \to \overline{P}^P(\vec{y}, \vec{y}) \in \Xi(\mathcal{P})$ and $\Xi(\mathcal{P}) \cup \mathcal{D} \models \xi_\top(\vec{a})$.

For the inductive step, assume that U contains at least two nodes and let $\{v_1, \ldots, v_n\}$ be the IDB leaves of U. Then, U has a node w of height 1. W.l.o.g., let $v_1, \ldots, v_k$ $(0 \leq k \leq n)$ be the IDB and $u_1, \ldots, u_l$ the EDB children of w in U and let $r = \varphi \wedge \bigwedge_{i=1}^{k} R_i(\vec{s}_i) \to \bigvee_{j=1}^{m} S_j(\vec{t}_j) \in \mathcal{P}$ (for some m) be the rule used in ρ to derive $\lambda(w)$ from $\lambda(v_1), \ldots, \lambda(v_k), \lambda(u_1), \ldots, \lambda(u_l)$ with substitution σ. We distinguish two cases:

- If $k = 0$, then we can observe the following:

  1. $U \backslash \{u_1, \ldots, u_l\}$ is an upper portion of T with IDB leaves $\{w, v_1, \ldots, v_n\}$;
  2. $\lambda(w) = \bigvee_{j=1}^{m} S_j(\vec{t}_j\sigma)$;
  3. $\varphi \wedge \bigwedge_{j=1}^{m} \overline{S}_j^P(\vec{t}_j, \vec{y}) \to P(\vec{y}) \in \Xi(\mathcal{P})$, by the definition of Ξ and the fact that $r \in \mathcal{P}$, where for $k = 0$ rule r is of the form $\varphi \to \bigvee_{j=1}^{m} S_j(\vec{t}_j)$;
  4. $\Xi(\mathcal{P}) \cup \mathcal{D} \models \varphi\sigma$ since $\mathcal{P} \cup \mathcal{D} \models \varphi\sigma$, $\varphi\sigma$ is a conjunction of EDB facts, and $\mathcal{P} \cup \mathcal{D}$ and $\Xi(\mathcal{P}) \cup \mathcal{D}$ entail the same EDB facts.

  By the first two observations and the inductive hypothesis, $\Xi(\mathcal{P}) \cup \mathcal{D}$ entails $P(\vec{a}) \vee \overline{S}_j^P(\vec{t}_j\sigma, \vec{a}) \vee \psi$ for every $S_j(\vec{t}_j\sigma) \in \lambda(w)$ and $\psi \in \Psi(v_1, \ldots, v_n)$. The claim follows by the third and fourth observations.

- If $k \geq 1$, then we can observe the following:

  1. $U \setminus \{v_1, \ldots, v_k, u_1, \ldots, u_l\}$ is an upper portion of T with IDB leaves $\{w, v_{k+1}, \ldots, v_n\}$;
  2. (a) $\bigvee_{j=1}^{m} S_j(\vec{t}_j\sigma) \subseteq \lambda(w)$ and (b) $\lambda(v_i) \subseteq \lambda(w) \cup \{R_i(\vec{s}_i\sigma)\}$ for each $i \in [1, k]$;
  3. $\xi_\top \wedge \varphi \wedge \bigwedge_{j=1}^{m} \overline{S}_j^P(\vec{t}_j, \vec{y}) \to \bigvee_{i=1}^{k} \overline{R}_i^P(\vec{s}_i, \vec{y}) \in \Xi(\mathcal{P})$;
  4. $\Xi(\mathcal{P}) \cup \mathcal{D} \models \xi_\top\sigma$ and $\Xi(\mathcal{P}) \cup \mathcal{D} \models \varphi\sigma$.

W.l.o.g., let $\Xi(\mathcal{P}) \cup \mathcal{D} \not\models P(\vec{a})$ (otherwise the claim is immediate), and let $\psi = \bigvee_{i=1}^{n} Q_i^P(\vec{b}_i, \vec{a}) \in \Psi(v_1, \ldots, v_n)$ be arbitrary. It suffices to show $\Xi(\mathcal{P}) \cup \mathcal{D} \models \psi$. By 1 and the inductive hypothesis, $\Xi(\mathcal{P}) \cup \mathcal{D} \models \overline{Q}^P(\vec{b}, \vec{a}) \vee \bigvee_{i=k+1}^{n} \overline{Q}_i^P(\vec{b}_i, \vec{a})$ for every $Q(\vec{b}) \in \lambda(w)$. Hence, by 2(b), the claim is immediate unless $Q_i(\vec{b}_i) = R_i(\vec{s}_i\sigma)$ for every $i \in [1, k]$, i.e., it suffices to show $\Xi(\mathcal{P}) \cup \mathcal{D} \models (\bigvee_{i=1}^{k} \overline{R}_i^P(\vec{s}_i\sigma, \vec{a})) \vee \bigvee_{i=k+1}^{n} \overline{Q}_i^P(\vec{b}_i, \vec{a})$. This follows by 3 and 4 since, by the inductive hypothesis and 2(a), $\Xi(\mathcal{P}) \cup \mathcal{D}$ entails $\overline{S}_j^P(\vec{t}_j\sigma, \vec{a}) \vee \bigvee_{i=k+1}^{n} \overline{Q}_i^P(\vec{b}_i, \vec{a})$ for every $j \in [1, m]$.

This concludes the proof of Claim ($\diamondsuit$).

We next use ($\diamondsuit$) to show $\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$. Let $\{v_1, \ldots, v_n\}$ be the IDB leaves of T. By ($\diamondsuit$), $\Xi(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a}) \vee \bigwedge \Psi(v_1, \ldots, v_n)$.[3] Thus, it suffices to show $\Xi(\mathcal{P}) \cup \mathcal{D} \cup \Psi(v_1, \ldots, v_n) \models P(\vec{a})$.

We show $\Xi(\mathcal{P}) \cup \mathcal{D} \cup \Psi(v_1, \ldots, v_i) \models P(\vec{a})$ for each $i \leq n$ by induction on i. If $i = 0$ (base case), the claim is immediate. For the inductive step, let $i \geq 1$. Then $\Psi(v_1, \ldots, v_i) = \{\, \psi \vee \overline{Q}^P(\vec{b}, \vec{a}) \mid \psi \in \Psi(v_1, \ldots, v_{i-1}),\ Q(\vec{b}) \in \lambda(v_i) \,\}$. By the inductive hypothesis, $\Xi(\mathcal{P}) \cup \mathcal{D} \cup \Psi(v_1, \ldots, v_{i-1}) \models P(\vec{a})$, and hence $\Xi(\mathcal{P}) \cup \mathcal{D} \cup \Psi(v_1, \ldots, v_i) \models P(\vec{a}) \vee \overline{Q}^P(\vec{b}, \vec{a})$ for each $Q(\vec{b}) \in \lambda(v_i)$. Hence, it suffices to show $\Xi(\mathcal{P}) \cup \mathcal{D} \cup \{\, \overline{Q}^P(\vec{b}, \vec{a}) \mid Q(\vec{b}) \in \lambda(v_i) \,\} \models P(\vec{a})$. For this, note that $\lambda(v_i)$ is a leaf in T obtained by a rule $(\rightarrow \lambda(v_i))$ where $\lambda(v_i) = \bigvee_{j=1}^{m} Q_j(\vec{b}_j)$. Then $\bigwedge_{j=1}^{m} \overline{Q}_j^P(\vec{b}_j, \vec{y}) \rightarrow P(\vec{y}) \in \Xi(\mathcal{P})$ and the claim follows.

Step 2. We show that $\Xi(\mathcal{P}) \cup \mathcal{D} \vdash P(\vec{a})$ implies $\mathcal{P} \cup \mathcal{D} \models P(\vec{a})$. For this, we first show that derivations from $\Xi(\mathcal{P}) \cup \mathcal{D}$ have a certain shape, which we define next. We call a disjunction of facts φ *focused on an IDB fact* $Q(\vec{b})$, with Q occurring in $\mathcal{P}$, if every disjunct in φ is either $Q(\vec{b})$ or a fact of the form $\overline{R}^Q(\vec{c}, \vec{b})$ for some R and $\vec{c}$. Let $\rho = (T, \lambda)$ be a derivation. We call ρ *focused on* $Q(\vec{b})$ if so is the label of each IDB node in T.

Claim ($\clubsuit$). Every derivation of a nonempty disjunction from $\Xi(\mathcal{P}) \cup \mathcal{D}$ is focused on some fact.

Let $\rho = (T, \lambda)$ be a derivation of a nonempty disjunction from $\Xi(\mathcal{P}) \cup \mathcal{D}$. We show that ρ is focused on some fact by induction on ρ. Let v be the root of T. For the base case, suppose v is the only node in T. We distinguish the following cases:

- $\lambda(v) \in \mathcal{D}$. Then, v is EDB and the claim is vacuous.
- $\mathsf{arity}(Q) = 0$ and $\lambda(v)$ is obtained by a rule of the form $(\rightarrow \overline{R}^Q) \in \Xi(\mathcal{P})$ for some IDB predicate Q in $\mathcal{P}$ and $R \in \{Q, \bot\}$. Then ρ is focused on Q.

[3]Note that if $\mathcal{P}$ has no rules with an empty body, all leaves of T are EDB and hence $n = 0$. Thus, the claim follows directly from ($\diamondsuit$).

For the inductive step, let $v_1, \ldots, v_m$ be the successors of v. We distinguish cases depending on the shape of the rule $r \in \Xi(\mathcal{P})$ used to derive $\lambda(v)$ from $\lambda(v_1), \ldots, \lambda(v_m)$.

- $r = \xi_\top \wedge \varphi \wedge \bigwedge_{j=1}^{k} \overline{R}_j^Q(\vec{t}_j, \vec{y}) \to \bigvee_{i=1}^{n} \overline{P}_i^Q(\vec{s}_i, \vec{y})$ ($n \geq 1$ and $k \leq m$). Let σ be the substitution used in the hyperresolution step. W.l.o.g., let $\overline{R}_j^Q(\vec{t}_j\sigma, \vec{y}\sigma) \in \lambda(v_j)$ for $j \in [1, k]$. Then $\lambda(v_j) \in \xi_\top \cup \varphi$ for $j \in [k+1, m]$. Moreover, by the inductive hypothesis, the subderivations rooted at $v_1, \ldots, v_k$ are focused on $Q(\vec{y}\sigma)$. The claim follows since we have $\lambda(v) \subseteq \lambda(v_1) \cup \cdots \cup \lambda(v_k) \cup \{\, \overline{P}_i^Q(\vec{s}_i\sigma, \vec{y}\sigma) \mid i \in [1, n] \,\}$.
- $r = \varphi \wedge \bigwedge_{j=1}^{k} \overline{R}_j^Q(\vec{t}_j, \vec{y}) \to Q(\vec{y})$. The argument is analogous to the preceding case.
- $r = \xi_\top \to \overline{R}^Q(\vec{t}, \vec{y})$ where $R \in \{Q, \bot\}$. Let σ be the substitution used in the hyperresolution step. Then ρ is focused on $Q(\vec{y}\sigma)$.

This concludes the proof of Claim (♣).

Let $\rho = (T, \lambda)$ be a derivation of $P(\vec{a})$ from $\Xi(\mathcal{P}) \cup \mathcal{D}$. By (♣), ρ is focused on $P(\vec{a})$ since the root w of ρ is labeled with $P(\vec{a})$. Given $v \in T$, we write $\lambda_{\text{base}}(v)$ to denote the set $\{\, Q(\vec{b}) \mid \overline{Q}^P(\vec{b}, \vec{a}) \in \lambda(v) \,\}$. We conclude our proof by showing the following property, which implies $\mathcal{P} \cup \mathcal{D} \models P(\vec{a})$ since $\lambda_{\text{base}}(w) = \emptyset$.

$$\mathcal{P} \cup \mathcal{D} \models (\bigwedge_{\alpha \in \lambda_{\text{base}}(v)} \alpha) \to P(\vec{a}) \quad \text{for every IDB node } v \text{ in } T$$

We show this property by induction on the height of v in T. Note that we only need to consider cases that can occur in a derivation focused on $P(\vec{a})$. For the base case, suppose v is a leaf in T. Since v is IDB, we have $\lambda(v) \notin \mathcal{D}$. Thus, we distinguish the following cases.

- $\lambda(v)$ is obtained by a rule of the form $\to \overline{\bot}^P$ where $\mathsf{arity}(P) = 0$. Then the claim reduces to $\mathcal{P} \cup \mathcal{D} \models \bot \to P$, which is tautological.
- $\lambda(v)$ is obtained by a rule of the form $\to \overline{P}^P$ where $\mathsf{arity}(P) = 0$. Then the claim reduces to $\mathcal{P} \cup \mathcal{D} \models P \to P$, which is tautological.

For the inductive step, let $v_1, \ldots, v_m$ be the successors of v in T. We distinguish cases depending on the shape of the rule $r \in \Xi(\mathcal{P})$ used to obtain $\lambda(v)$ from $\lambda(v_1), \ldots, \lambda(v_m)$:

- $r = \xi_\top \wedge \varphi \wedge \bigwedge_{j=1}^{k} \overline{Q}_j^P(\vec{t}_j, \vec{y}) \to \bigvee_{i=1}^{n} \overline{R}_i^P(\vec{s}_i, \vec{y})$ such that $k \leq m$, $n \geq 1$ and $r' = \varphi \wedge \bigwedge_{i=1}^{n} R_i(\vec{s}_i) \to \bigvee_{j=1}^{k} Q_j(\vec{t}_j) \in \mathcal{P}$. Let σ be the substitution used in the corresponding hyperresolution step and let, w.l.o.g., $\overline{Q}_j^P(\vec{t}_j\sigma, \vec{y}\sigma) \in \lambda(v_j)$ for each $j \in [1, k]$. By the inductive hypothesis, $\mathcal{P} \cup \mathcal{D} \models (\bigwedge_{\alpha \in \lambda_{\text{base}}(v_j)} \alpha) \to P(\vec{a})$ for $j \in [1, k]$. Moreover, we have $\bigcup_{j=1}^{k} (\lambda_{\text{base}}(v_j) \setminus \{Q_j(\vec{t}_j\sigma)\}) \cup \bigcup_{i=1}^{n} \{R_i(\vec{s}_i\sigma)\} \subseteq \lambda_{\text{base}}(v)$. The claim follows since $\mathcal{P} \cup \mathcal{D} \models \varphi\sigma$ and hence $\mathcal{P} \cup \mathcal{D} \models (\bigwedge_{\alpha \in \lambda_{\text{base}}(v)} \alpha) \to \bigvee_{j=1}^{k} Q_j(\vec{t}_j\sigma)$ by rule r'.

- $r = \varphi \wedge \bigwedge_{j=1}^{k} \overline{Q}_j^P(\vec{t}_j, \vec{y}) \to P(\vec{y})$ such that $r' = \varphi \to \bigvee_{j=1}^{k} Q(\vec{t}) \in \mathcal{P}$. Let σ be the relevant substitution and $\overline{Q}_j^P(\vec{t}_j\sigma, \vec{y}\sigma) \in \lambda(v_j)$ for each $j \in [1, k]$. By induction, $\mathcal{P} \cup \mathcal{D} \models (\bigwedge_{\alpha \in \lambda_{\text{base}}(v_j)} \alpha) \to P(\vec{a})$. Moreover, we have $\bigcup_{j=1}^{k}(\lambda_{\text{base}}(v_j) \setminus \{Q_j(\vec{t}_j\sigma)\}) \subseteq \lambda_{\text{base}}(v)$. The claim holds since $\mathcal{P} \cup \mathcal{D} \models \varphi\sigma$ and hence, by rule r', $\mathcal{P} \cup \mathcal{D} \models (\bigwedge_{\alpha \in \lambda_{\text{base}}(v)} \alpha) \to \bigvee_{j=1}^{k} Q_j(\vec{t}_j\sigma)$.

- $r = \xi_\top \to \overline{\bot}^P(\vec{y})$. Since ρ is focused on $P(\vec{a})$, the claim reduces to the tautology $\mathcal{P} \cup \mathcal{D} \models \bot \to P(\vec{a})$.

- $r = \xi_\top \to \overline{P}^P(\vec{y}, \vec{y})$. Since ρ is focused on $P(\vec{a})$, the claim reduces to the tautology $\mathcal{P} \cup \mathcal{D} \models P(\vec{a}) \to P(\vec{a})$.

This completes the proof of Theorem 3. □

3.3. Correspondence between Datalog and Linear Disjunctive Datalog

Theorem 3 and Proposition 2 have the following direct implication. On the one hand, if $\mathcal{P}$ is a linear Disjunctive Datalog program, then its transposition results in a polynomial-size Datalog rewriting of $\mathcal{P}$. On the other hand, if $\mathcal{P}$ is Datalog, then its transposition is a polynomial-size rewriting of $\mathcal{P}$ in linear Disjunctive Datalog.

A characterisation of Datalog rewritability immediately becomes apparent: a Disjunctive Datalog program is Datalog rewritable if and only if it is linearisable. But not only that, the fact that program transposition is a polynomial transformation also means that linear Disjunctive Datalog enjoys the same favourable computational properties as plain Datalog, such as tractability of reasoning in data complexity.

Theorem 4. *A Disjunctive Datalog program $\mathcal{P}$ is Datalog rewritable if and only if it is rewritable into linear Disjunctive Datalog. Moreover, fact entailment w.r.t. linear Disjunctive Datalog programs is* ExpTime*-complete in combined complexity and* PTime*-complete w.r.t. data.*

4. Markable Programs

We have established that every linear Disjunctive Datalog program can be polynomially rewritten into Datalog. It is, however, rather straightforward to come up with disjunctive programs that are syntactically non-linear, but also admit a polynomial Datalog rewriting. In this section, we introduce *markable programs*: a larger class of Disjunctive Datalog programs that admit polynomial Datalog rewritings by means of a generalisation of transposition.

To illustrate the main ideas behind markable programs, we will start by first introducing in Section 4.1 a more restricted (but also more intuitive) class of *weakly linear* (WL) programs. We will then exploit these intuitions to motivate in Section 4.2 the notions of *marking* and *markable program.* In Section 4.3 we show that markable programs can be efficiently recognised by means of a reduction into 2-SAT. Finally, in Sections 4.4 and 4.5 we show how transposition can be extended in order to polynomially rewrite markable programs into Datalog.

4.1. Weakly Linear Programs

Consider the following program $\mathcal{P}_3$, which extends our example program $\mathcal{P}_1$ in Section 3 with the following rules:

$$A(x,y) \to E(x,y) \qquad (8) \qquad\qquad E(y,x) \to E(x,y) \qquad (9)$$

Since E is IDB in $\mathcal{P}_3$, Rules (2) and (3) have two IDB atoms, which breaks the linearity condition. The main idea behind weak linearity is simple: instead of requiring rule bodies to contain at most one occurrence of an IDB predicate, we require at most one occurrence of a *disjunctive* predicate—a predicate whose extension for some dataset could depend on the application of a disjunctive rule. In particular, the extension of predicate E in $\mathcal{P}_3$ depends only on the newly introduced Datalog rules (8) and (9), and hence E would not be disjunctive.

This intuition is formalised as follows using the standard notion of a dependency graph in Logic Programming [20, 21].

Definition 5. The *dependency graph* of a program $\mathcal{P}$ is the smallest edge-labeled digraph $G_{\mathcal{P}} = (V, E, \mu)$ such that

1. V contains every predicate occurring in $\mathcal{P}$;
2. $r \in \mu(P,Q)$ whenever $P, Q \in V$, $r \in \mathcal{P}$, P occurs in the body of r, and Q occurs in the head of r; and
3. $(P,Q) \in E$ whenever $\mu(P,Q)$ is nonempty.

A predicate Q *depends on a rule* $r \in \mathcal{P}$ if $G_{\mathcal{P}}$ has a path that ends in Q and involves an r-labeled edge. Predicate Q is *Horn* if it only depends on Horn rules; otherwise, Q is *disjunctive*. An atom is Horn (resp., disjunctive) if so is its predicate. A program is *weakly linear* (WL) if each of its rules has at most one disjunctive body atom. $\diamond$

Note that, if $\mathcal{P}$ is Datalog, then all its predicates are Horn and $\mathcal{P}$ is WL; furthermore, every disjunctive predicate is IDB and hence every linear Disjunctive Datalog program is also WL. Thus, the class of WL programs extends both Datalog and linear Disjunctive Datalog.

The dependency graph $G_{\mathcal{P}_3}$ of our example program $\mathcal{P}_3$ is then as given next. We can observe that predicates A and C are EDB and hence do not depend on any rule. As already mentioned, predicate E depends only on Datalog rules and hence is Horn. Finally, B and G depend on Rule (1) and hence are disjunctive. Consequently, $\mathcal{P}_3$ is WL.

$$\begin{tikzcd}
& B \arrow[dd, bend left, "(3)"] & & \\
C \arrow[ur, "(1)"] \arrow[dr, "(1)"'] & & E \arrow[ul, "(2)"'] \arrow[dl, "(3)"] \arrow[loop below, "(9)"] & A \arrow[l, "(8)"'] \\
& G \arrow[uu, bend left, "(2)"] & &
\end{tikzcd}$$

Transposition can be easily adapted to polynomially rewrite WL programs into plain Datalog. Rather than moving all IDB atoms between head and body, it suffices to move only disjunctive atoms. In this way, rules that contain only Horn predicates remain unaffected by the transformation. In particular, the refined transposition of our example program $\mathcal{P}_3$ would consist of the following rules, where, once again, each rule mentioning X stands for one rule where $X = B$ and one where $X = G$:

$$\begin{aligned}
\Xi'(\mathcal{P}_3) = \{\, & C(x) \wedge \overline{B}^X(x,z) \wedge \overline{G}^X(x,z) \rightarrow X(z), & \text{(1')}\\
& \overline{B}^X(x,z) \wedge E(x,y) \rightarrow \overline{G}^X(y,z), & \text{(2')}\\
& \overline{G}^X(x,z) \wedge E(x,y) \rightarrow \overline{B}^X(y,z), & \text{(3')}\\
& \top(x) \rightarrow \overline{X}^X(x,x), & \text{(4)}\\
& A(x,y) \rightarrow E(x,y), & \text{(8)}\\
& E(y,x) \rightarrow E(x,y)\,\} & \text{(9)}
\end{aligned}$$

Note that Rules (1')–(4) are the same as in $\Xi(\mathcal{P}_1)$ whereas Rules (8) and (9) are copied from $\mathcal{P}_3$ since they contain only Horn predicates.

4.2. Markings and Markable Programs

The class of markable programs is obtained by further relaxing the weak linearity requirement to apply only to a subset of *marked* disjunctive predicates in a program $\mathcal{P}$. These predicates, however, must be chosen in such a way that the transposition of $\mathcal{P}$ where only marked atoms are transposed between head and body results in a Horn program. A program can admit many different markings, and markable programs are those that admit at least one marking.

Definition 6. A *marking* of a program $\mathcal{P}$ is a set M of disjunctive predicates in $\mathcal{P}$ with the following properties, where we say that a predicate is marked if it is in M and an atom is marked if so is its predicate:

1. each rule in $\mathcal{P}$ has at most one marked body atom;
2. each rule in $\mathcal{P}$ has at most one unmarked head atom; and
3. if Q is marked and P is reachable from Q in $G_{\mathcal{P}}$, then P is marked.

We say that a program is *markable* if it admits a marking. ◇

Condition 1 in Definition 6 ensures that at most one atom is moved from body to head during transposition. Condition 2 ensures that all but possibly one head atom are moved to the body. Finally, Condition 3 requires that all predicates depending on a marked predicate are also marked. Markability generalises weak linearity: a program is WL if and only if the set of all its disjunctive predicates constitutes a marking. Consider now program $\mathcal{P}_4$, which extends our example program $\mathcal{P}_3$ with the following rules:

$$V(x) \rightarrow C(x) \vee U(x) \qquad (10) \qquad\qquad C(x) \wedge U(x) \rightarrow \bot \qquad (11)$$

The dependency graph is given next. Note that C, U, B, G, and $\bot$ are disjunctive as they depend on Rule (10). Thus, (11) has two disjunctive body atoms and $\mathcal{P}_4$ is not WL. The program, however, has two different markings: $\{C, B, G, \bot\}$ and $\{U, B, G, \bot\}$.

$$\begin{tikzcd}
& V \arrow[ld, "(10)"'] \arrow[rd, "(10)"] & & B \arrow[d, bend left, "(3)"] & & \\
U \arrow[rd, "(11)"'] & & C \arrow[ld, "(11)"] \arrow[ru, "(1)"] \arrow[rd, "(1)"'] & & E \arrow[lu, "(2)"'] \arrow[ld, "(3)"] \arrow[loop below, "(9)"] & A \arrow[l, "(8)"'] \\
& \bot & & G \arrow[u, bend left, "(2)"] & &
\end{tikzcd}$$

4.3. Checking Markability

We next show that markability can be efficiently checked via a quadratic reduction to 2-SAT. The reduction assigns to each predicate Q in $\mathcal{P}$ a distinct propositional variable X_Q; it then encodes directly the constraints in Definition 6 as binary clauses. Furthermore, our reduction provides tight complexity bounds for markability checking since 2-SAT is NLogSpace-complete.

Proposition 7. *Markability is* NLogSpace*-complete and can be checked in time quadratic in the size of the input $\mathcal{P}$.*

Proof. For each rule in $\mathcal{P}$ of the form $\varphi \wedge \bigwedge_{i=1}^{n} P_i(\vec{s}_i) \to \bigvee_{j=1}^{m} Q_j(\vec{t}_j)$, where φ is the conjunction of all its Horn atoms, we obtain the following clauses:

1. $\neg X_{P_i} \vee \neg X_{P_j}$ for all $1 \leq i < j \leq n$;
2. $\neg X_{P_i} \vee X_{Q_j}$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$; and
3. $X_{Q_i} \vee X_{Q_j}$ for all $1 \leq i < j \leq m$.

Clauses of the form (1) indicate that at most one body atom may be marked. By (2), if a body atom is marked, then so must be all head atoms. Finally, (3) ensures that at most one head atom may be unmarked. The resulting set $\mathcal{N}$ of propositional clauses is satisfiable iff $\mathcal{P}$ has a marking, and every model I of $\mathcal{N}$ yields a marking. Furthermore, the reduction is clearly feasible in LogSpace and hence markability is in NLogSpace. Finally, $|\mathcal{N}|$ is quadratic in $|\mathcal{P}|$ and hence markability can be checked in quadratic time (2-SAT is feasible in linear time). To establish NLogSpace-hardness, we argue that any 2-SAT instance $\mathcal{N}$ can be reduced to markability of a program $\mathcal{P}$. For this, we define $\mathcal{P}$ as a (propositional) Disjunctive Datalog program consisting of the following rules:

- $\to P_p \vee P_q$ for every clause $p \vee q \in \mathcal{N}$;
- $P_p \to P_q$ for every clause $\neg p \vee q \in \mathcal{N}$;
- $P_p \wedge P_q \to P$ for every clause $\neg p \vee \neg q \in \mathcal{N}$, where P is a fresh predicate.

It is easily seen that $\mathcal{P}$ has a marking iff $\mathcal{N}$ is satisfiable. $\square$

4.4. Rewriting Markable Disjunctive Datalog Programs into Datalog

We finally show how transposition can be refined in order to polynomially rewrite markable programs into Datalog.

Consider program $\mathcal{P}_4$ and the marking $M = \{U, B, G, \bot\}$. The transposition of $\mathcal{P}_4$ will depend on the particular marking under consideration, so let us denote the M-transposed program as $\Xi_M(\mathcal{P}_4)$. We introduce fresh binary predicates $\overline{X}^Y$, where X and Y are disjunctive and X is required to be marked. Such predicates $\overline{X}^Y$ carry the same intended meaning as in the original notion of transposition: if a fact $\overline{B}^G(c, d)$ holds in $\Xi_M(\mathcal{P}_4) \cup \mathcal{D}$ then proving $B(c)$ suffices for proving $G(d)$ in $\mathcal{P}_4 \cup \mathcal{D}$ (i.e., $\mathcal{P}_4 \cup \mathcal{D} \models B(c) \rightarrow G(d)$). The extension of these predicates is again initialised with rules $\top(x) \rightarrow \overline{X}^X(x, x)$ for each $X \in M$.

For each unmarked disjunctive predicate X and each disjunctive predicate Y (marked or not) we now introduce a fresh predicate X^Y that comes with a different intuitive interpretation: if a fact $C^U(c, d)$ holds in $\Xi_M(\mathcal{P}_4) \cup \mathcal{D}$ then $\mathcal{P}_4 \cup \mathcal{D}$ entails $C(c) \vee U(d)$.

As usual in transposition, the key step is to transpose atoms between head and body while replacing their predicates with auxiliary ones. Now, however, we do this only for those rules that involve the marked predicates B, G and U. For example, Rule (2) leads to the following rules in $\Xi_M(\mathcal{P}_4)$, for each unary disjunctive predicate Y as well as for $\bot$:

$$\overline{B}^Y(x, z) \wedge E(x, y) \rightarrow \overline{G}^Y(y, z) \qquad \overline{B}^\bot(x) \wedge E(x, y) \rightarrow \overline{G}^\bot(y)$$

Unmarked disjunctive atoms are also modified by replacing their predicate with an auxiliary one; however, in contrast to marked atoms, they are not moved from one side of the rule to the other. Thus, Rules (10) and (11) yield the following rules for the $\bot$ predicate and each unary disjunctive predicate Y:

$$\begin{array}{ll} V(x) \wedge \overline{U}^Y(x, y) \rightarrow C^Y(x, y) & C^Y(x, y) \wedge \overline{\bot}^Y(y) \rightarrow \overline{U}^Y(x, y) \\ V(x) \wedge \overline{U}^\bot(x) \rightarrow C^\bot(x) & C^\bot(x) \wedge \overline{\bot}^\bot \rightarrow \overline{U}^\bot(x) \end{array}$$

Indeed, these rules are consequences of (10) and (11), respectively, under the intended meaning of the auxiliary predicates corresponding to unmarked disjunctive predicates: $V(a)$ and $U(a) \rightarrow Y(b)$ imply $C(a) \vee Y(b)$ by (10), while $C(a) \vee Y(b)$, $\bot \rightarrow Y(b)$ (a tautology), and $U(a)$ imply $Y(b)$ by (11).

We are now ready to define our transformation. As in the previous case, the transformation is quadratic and the arity of predicates is at most doubled.

Definition 8. Let $\mathcal{P}$ be a Disjunctive Datalog program and let M be a marking of $\mathcal{P}$. For each pair (P, Q) of disjunctive predicates in $\mathcal{P}$, let P^Q and $\overline{P}^Q$ be fresh predicates of arity $\mathsf{arity}(P) + \mathsf{arity}(Q)$. The M-transposition of $\mathcal{P}$ is the smallest program $\Xi_M(\mathcal{P})$ containing each rule in $\mathcal{P}$ with no disjunctive predicates and all Rules 1–5 given next, where $\xi_\top$ is the least conjunction of $\top$-atoms needed to make a rule safe, φ is the conjunction of all Horn atoms in a rule, predicates P_i, Q_j are disjunctive, and $\vec{y}, \vec{z}$ are vectors of distinct fresh variables:

1. $\xi_\top \rightarrow \overline{R}^R(\vec{y}, \vec{y})$ for each disjunctive predicate $R \in M$;

2. $\xi_\top \wedge \varphi \wedge \bigwedge_j Q_j^R(\vec{t}_j, \vec{y}) \wedge \bigwedge_i \overline{P}_i^R(\vec{s}_i, \vec{y}) \to \overline{Q}^R(\vec{t}, \vec{y})$ for each rule $r \in \mathcal{P}$ of the form $\varphi \wedge Q(\vec{t}) \wedge \bigwedge_j Q_j(\vec{t}_j) \to \bigvee_i P_i(\vec{s}_i)$ and each disjunctive predicate R in $\mathcal{P}$, where $Q(\vec{t})$ is the (unique) marked body atom of r;

3. $\varphi \wedge \bigwedge_j Q_j^R(\vec{t}_j, \vec{y}) \wedge \bigwedge_i \overline{P}_i^R(\vec{s}_i, \vec{y}) \to R(\vec{y})$ for each rule $r \in \mathcal{P}$ of the form $\varphi \wedge \bigwedge_j Q_j(\vec{t}_j) \to \bigvee_i P_i(\vec{s}_i)$ and each disjunctive predicate R in $\mathcal{P}$, where r has no marked body atoms and no unmarked head atoms;

4. $\xi_\top \wedge \varphi \wedge \bigwedge_j Q_j^R(\vec{t}_j, \vec{y}) \wedge \bigwedge_i \overline{P}_i^R(\vec{s}_i, \vec{y}) \to P^R(\vec{s}, \vec{y})$ for each rule $r \in \mathcal{P}$ of the form $\varphi \wedge \bigwedge_j Q_j(\vec{t}_j) \to P(\vec{s}) \vee \bigvee_i P_i(\vec{s}_i)$ and each disjunctive predicate R, where r has no marked body atoms and $P(\vec{s})$ is the only unmarked head atom; and

5. $R^R(\vec{y}, \vec{y}) \to R(\vec{y})$ for each disjunctive predicate $R \notin M$. ◇

Rules 1–3 are analogous to those in Definition 1, with the difference that rules of type 1 are generated only for marked predicates, different auxiliary predicates are used depending on whether the relevant base predicates are marked or not, and only marked atoms are moved in rules of type 2 and 3. Rules of type 4 are analogous to those of type 2, but deal with the case where the relevant rule in $\mathcal{P}$ contains no marked body atom. Finally, rules of type 5 ensure that unmarked predicates R are instantiated with constants $\vec{a}$ whenever rules of type 4 yield facts $R^R(\vec{a}, \vec{a})$; thus, these rules encode a tautology under the intended meaning of the auxiliary predicates R^R.

4.5. Correctness of the Rewriting

The rest of this section is devoted to showing that $\Xi_M(\mathcal{P})$ is indeed a Datalog rewriting of a Disjunctive Datalog program $\mathcal{P}$ whenever M is a marking of $\mathcal{P}$. The correctness argument is similar in structure to the proof of Theorem 3 but it is more involved, as it requires a separate treatment for the auxiliary predicates depending on whether or not the relevant base predicate is marked.

Theorem 9. *Let $\mathcal{P}$ be a Disjunctive Datalog program and let M be a marking of $\mathcal{P}$. Then $\Xi_M(\mathcal{P})$ is a polynomial-size Datalog rewriting of $\mathcal{P}$.*

Proof. As in the proof of Theorem 3, we proceed in two steps, which together imply the theorem. We fix an arbitrary markable program $\mathcal{P}$, a marking M of $\mathcal{P}$, a dataset $\mathcal{D}$ over the EDB predicates of $\mathcal{P}$, and a fact $P(\vec{a})$ with P disjunctive in $\mathcal{P}$ (if P is Horn the claim is immediate).

1. In Step 1, we show that $\mathcal{P} \cup \mathcal{D} \vdash P(\vec{a})$ implies $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$. To this end, we consider a derivation ρ of $P(\vec{a})$ from $\mathcal{P} \cup \mathcal{D}$ and show that for every disjunctive atom $Q(\vec{b})$ in the label of a node in ρ, we have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{Q}^P(\vec{b}, \vec{a})$ if $Q \in M$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q^P(\vec{b}, \vec{a})$ otherwise. This claim, in turn, is shown by first showing a more general statement and then instantiating it with ρ.

2. In Step 2, we show that $\Xi_M(\mathcal{P}) \cup \mathcal{D} \vdash P(\vec{a})$ implies $\mathcal{P} \cup \mathcal{D} \models P(\vec{a})$. Again, we first prove a general claim that holds for any derivation from $\Xi_M(\mathcal{P}) \cup \mathcal{D}$ and then instantiate the claim with a derivation of $P(\vec{a})$.

In both steps we use that $\mathcal{P}$ and $\Xi_M(\mathcal{P})$ entail the same facts over Horn predicates for every dataset. As in the proof of Theorem 3, the statement in the theorem follows from these steps and the completeness of hyperresolution.

Step 1. Suppose $\mathcal{P} \cup \mathcal{D} \vdash P(\vec{a})$. We show $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$. We begin by proving the following claim.

Claim ($\diamondsuit$). Let $\varphi = Q_1(\vec{b}_1) \vee \cdots \vee Q_n(\vec{b}_n)$ be a non-empty disjunction of facts satisfying the following properties: *(i)* $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{Q}_i^P(\vec{b}_i, \vec{a})$ for each $Q_i \in M$. *(ii)* φ is derivable from $\mathcal{P} \cup \mathcal{D}$. Then, for each derivation ρ of φ from $\mathcal{P} \cup \mathcal{D}$ and each atom $R(\vec{c})$ with R disjunctive in the label of a node in ρ we have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{R}^P(\vec{c}, \vec{a})$ if $R \in M$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models R^P(\vec{c}, \vec{a})$ otherwise.

The claim is proved by induction on $\rho = (T, \lambda)$. W.l.o.g., the root v of T has a disjunctive predicate in its label (otherwise, the claim is vacuous). Since disjunctive predicates are IDB, we have $\varphi \notin \mathcal{D}$ and hence it is obtained by a rule application.

For the base case, suppose v has no children labeled with disjunctive predicates. Thus, φ is obtained by a rule $\psi \to \varphi' \in \mathcal{P}$ where ψ is a conjunction of Horn atoms and, for some σ, $\varphi = \varphi'\sigma$ and $\mathcal{P} \cup \mathcal{D} \models \psi\sigma$. If $\{Q_1, \ldots, Q_n\} \subseteq M$, then the claim is immediate by assumption *(i)*, so let us assume w.l.o.g. that $Q_1 \notin M$. By the definition of a marking, we then have $\{Q_2, \ldots, Q_n\} \subseteq M$, and hence it suffices to show $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_1^P(\vec{b}_1, \vec{a})$. For this, note that $\psi \to \varphi' \in \mathcal{P}$, $\{Q_2, \ldots, Q_n\} \subseteq M$, and $Q_1 \notin M$ implies $r' = \xi_\top \wedge \psi \wedge \bigwedge_{i=2}^n \overline{Q}_i^P(\vec{x}_i, \vec{y}) \to Q_1^P(\vec{x}_1, \vec{y}) \in \Xi_M(\mathcal{P})$ for some variables $\vec{x}_i$ such that $\vec{x}_i = \vec{b}_i$. Moreover, by assumption *(i)*, we have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \bigwedge_{i=2}^n \overline{Q}_i^P(\vec{b}_i, \vec{a})$. Finally, since $\Xi_M(\mathcal{P}) \cup \mathcal{D}$ and $\mathcal{P} \cup \mathcal{D}$ entail the same Horn facts, we have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \psi\sigma$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \xi_\top\sigma$. Thus, the claim follows with r'.

For the inductive step, let v have children $w_1, \ldots, w_m$ in T labeled with disjunctive predicates. W.l.o.g., there is a rule $r = \psi \wedge \bigwedge_{i=1}^m R_i(\vec{t}_i) \to \bigvee_{j=1}^k Q_j(\vec{s}_j)$ in $\mathcal{P}$ (with ψ a conjunction of Horn atoms, $0 \leq k \leq n$, and all R_i disjunctive in $\mathcal{P}$) such that $\lambda(v)$ is obtained by a hyperresolution step using r from $\psi\sigma$ and $\lambda(w_1), \ldots, \lambda(w_m)$ where σ is a substitution mapping every atom $R_i(\vec{t}_i)$ to a disjunct in $\lambda(w_i)$. In particular, we have $\vec{s}_j\sigma = \vec{b}_j$, $R_i(\vec{t}_i\sigma) \in \lambda(w_i)$, and $\mathcal{P} \cup \mathcal{D} \models \psi\sigma$. We distinguish three cases:

- $\{Q_1, \ldots, Q_k\} \subseteq M$ and $\{R_1, \ldots, R_m\} \cap M = \emptyset$. For each $i \in [1, m]$, every marked atom in $\lambda(w_i)$ also occurs in $\lambda(v)$; furthermore, every unmarked atom in $\lambda(v)$ occurs in $\lambda(w_i)$ for some $i \in [1, m]$. By the latter statement, it suffices to show the claim for the subderivations rooted at $w_1, \ldots, w_m$.

  Let $i \in [1, m]$. By the fact that every marked atom in $\lambda(w_i)$ also occurs in $\lambda(v)$ and assumption *(i)*, we have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{S}^P(\vec{d}, \vec{a})$ for every marked disjunct $S(\vec{d})$ in $\lambda(w_i)$. Then, we can apply the inductive hypothesis to the subderivation rooted at w_i and the claim follows.

- $\{Q_1,\ldots,Q_k\} \subseteq M$, $R_1 \in M$, and $\{R_2,\ldots,R_m\} \cap M = \emptyset$ (note that $R_1 \in M$ implies $\{R_2,\ldots,R_m\} \cap M = \emptyset$ since M is a marking). Then (a) for every $i \in [1,m]$, every marked atom in $\lambda(w_i)$ except for possibly $R_1(t_1\sigma)$ in $\lambda(w_1)$ also occurs in $\lambda(v)$, and (b) every unmarked atom in $\lambda(v)$ occurs in $\lambda(w_i)$ for some $i \in [1,m]$. Also, we have (c) $\xi_\top \wedge \psi \wedge \bigwedge_{i=2}^m R_i^P(\vec{t}_i,\vec{y}) \wedge \bigwedge_{j=1}^k \overline{Q}_j^P(\vec{s}_j,\vec{y}) \to \overline{R}_1^P(\vec{t}_1,\vec{y}) \in \Xi_M(\mathcal{P})$. As in the preceding case, by (b), it suffices to show the claim for the subderivations rooted at $w_1,\ldots,w_m$. For $w_2,\ldots,w_n$, we proceed as follows. Let $i \in [2,m]$. By (a) and assumption *(i)*, we have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{S}^P(\vec{d},\vec{a})$ for every marked disjunct $S(\vec{d})$ in $\lambda(w_i)$. Thus, we can apply the inductive hypothesis to the subderivation rooted at w_i. In particular, we obtain $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models R_i^P(\vec{t}_i\sigma,\vec{a})$. In the case of w_1, we need to show $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{R}_1^P(\vec{t}_1\sigma,\vec{a})$ in order to apply the inductive hypothesis. This follows by (c) and assumption *(i)* since $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \psi\sigma$, $\{Q_1(\vec{s}_1\sigma),\ldots,Q_k(\vec{s}_k\sigma)\} \subseteq \lambda(v)$, $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models R_i^P(\vec{t}_i\sigma,\vec{a})$ for $i \in [2,m]$, and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \xi_\top\sigma$.

- $Q_1 \notin M$, $\{Q_2,\ldots,Q_k\} \subseteq M$, and $\{R_1,\ldots,R_m\} \cap M = \emptyset$ (note that $Q_1 \notin M$ implies $\{Q_2,\ldots,Q_k\} \subseteq M$ and $\{R_1,\ldots,R_m\} \cap M = \emptyset$). Then (a) for every $i \in [1,m]$, every marked atom in $\lambda(w_i)$ also occurs in $\lambda(v)$, and (b) every unmarked atom in $\lambda(v)$ except for possibly $Q_1(\vec{b}_1)$ (but including $Q_2(b_2),\ldots,Q_m(b_m)$) occurs in $\lambda(w_i)$ for some $i \in [1,m]$. By (b), it suffices to show the main claim for the subderivations rooted at $w_1,\ldots,w_m$ and also that $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_1^P(\vec{b}_1,\vec{a})$. Let $i \in [1,m]$. The main claim for the subderivations follows from (a) and assumption *(i)*, which imply that $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{S}^P(\vec{d},\vec{a})$ for every marked disjunct $S(\vec{d})$ in $\lambda(w_i)$; as a result, we can apply the inductive hypothesis to the subderivation rooted at w_i. Finally, $\xi_\top \wedge \psi \wedge \bigwedge_{i=1}^m R_i^P(\vec{t}_i,\vec{y}) \wedge \bigwedge_{j=2}^k \overline{Q}_j^P(\vec{s}_j,\vec{y}) \to Q_1^P(\vec{s}_1,\vec{y}) \in \Xi_M(\mathcal{P})$ (since $r \in \mathcal{P}$). Then, $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_1^P(\vec{b}_1,\vec{a})$ follows from $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \psi\sigma$, the inductive hypothesis (which implies $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models R_i^P(\vec{t}_i\sigma,\vec{a})$), and the assumption *(i)* (which implies $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{Q}_j^P(\vec{s}_j\sigma,\vec{a})$).

This concludes the proof of Claim ($\diamondsuit$).

We next instantiate ($\diamondsuit$) to show the claim in Step 1. Let $\varphi = P(\vec{a})$. We have assumed in Step 1 that $P(\vec{a})$ is derivable from $\mathcal{P} \cup \mathcal{D}$, and hence condition *(ii)* in ($\diamondsuit$) holds. Furthermore, $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{P}^P(\vec{a},\vec{a})$ since $\xi_\top(\vec{y}) \to \overline{P}^P(\vec{y},\vec{y}) \in \Xi_M(\mathcal{P})$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \xi_\top(\vec{a})$; hence, condition *(i)* in ($\diamondsuit$) also holds.

Now, let $\rho = (T,\lambda)$ be a derivation of $P(\vec{a})$ from $\mathcal{P} \cup \mathcal{D}$. We exploit ($\diamondsuit$) applied to ρ to show that $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models P(\vec{a})$. We distinguish two cases:

- $P \notin M$. Since $P(\vec{a})$ labels the root of ρ we can apply ($\diamondsuit$) to obtain $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models P^P(\vec{a},\vec{a})$; the claim follows since $P^P(\vec{y},\vec{y}) \to P(\vec{y}) \in \Xi_M(\mathcal{P})$.

- $P \in M$. Then, there is an IDB node v in ρ such that: $\lambda(v)$ contains only marked atoms and v has no successor w in T such that all atoms in $\lambda(w)$ are marked. Since v is IDB, it has successors $v_1,\ldots,v_n$ ($n \geq 0$) in T such that $\lambda(v)$ is a hyperresolvent of $\lambda(v_1),\ldots,\lambda(v_n)$ and a rule in $\mathcal{P}$ of the form $\bigwedge_{i=1}^{n} Q_i(\vec{s}_i) \to \bigvee_{j=1}^{m} R_j(\vec{t}_j)$, where the atoms $Q_i(\vec{s}_i)$ are resolved with $\lambda(v_i)$. Since $\lambda(v)$ contains only marked atoms but $\lambda(v_1), \ldots, \lambda(v_n)$ all contain Horn or unmarked atoms, all Q_i must be Horn or unmarked and all R_j must be marked. Hence, $\Xi_M(\mathcal{P})$ contains a rule $r = (\bigwedge_{i=1}^{k} Q_i(\vec{s}_i)) \wedge (\bigwedge_{l=k+1}^{n} Q_l^P(\vec{s}_l, \vec{y})) \wedge \bigwedge_{j=1}^{m} \overline{R}_j^P(\vec{t}_j, \vec{y}) \to P(\vec{y})$ where, w.l.o.g., $Q_1, \ldots, Q_k$ are Horn and $Q_{k+1}, \ldots, Q_n$ are disjunctive and unmarked. Let σ be the substitution used in the hyperresolution step deriving $\lambda(v)$. By ($\diamondsuit$), we then have $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_l^P(\vec{s}_l\sigma, \vec{a})$ for every $l \in [k+1, n]$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{R}_j^P(\vec{t}_j\sigma, \vec{a})$ for every $j \in [1, m]$. Moreover, we have $\lambda(v_i) = Q_i(\vec{s}_i\sigma)$ and hence $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_i(\vec{s}_i\sigma)$ for every $i \in [1, k]$. The claim follows with r.

Step 2. Let $\Xi_M(\mathcal{P}) \cup \mathcal{D} \vdash P(\vec{a})$ with ρ a derivation of $P(\vec{a})$ from $\Xi_M(\mathcal{P}) \cup \mathcal{D}$. The fact that $\mathcal{P} \cup \mathcal{D} \models P(\vec{a})$ follows directly from Statement 2 in Claim ($\clubsuit$), which we show next.

Claim ($\clubsuit$). Let ρ be a derivation from $\Xi_M(\mathcal{P}) \cup \mathcal{D}$ with root v. Then:

1. The label $\lambda(v)$ of v has one of the following forms:
   - $Q(\vec{b})$ where Q occurs in $\mathcal{P}$,
   - $\overline{Q}^R(\vec{b}, \vec{c})$ where Q, R are disjunctive in $\mathcal{P}$ and $Q \in M$; and
   - $Q^R(\vec{b}, \vec{c})$ where Q, R are disjunctive in $\mathcal{P}$ and $Q \notin M$.
2. If $\lambda(v) = Q(\vec{b})$ where Q occurs in $\mathcal{P}$, then $\mathcal{P} \cup \mathcal{D} \models Q(\vec{b})$.
3. If $\lambda(v) = \overline{Q}^R(\vec{b}, \vec{c})$, then $\mathcal{P} \cup \mathcal{D} \models Q(\vec{b}) \to R(\vec{c})$.
4. If $\lambda(v) = Q^R(\vec{b}, \vec{c})$, then $\mathcal{P} \cup \mathcal{D} \models Q(\vec{b}) \vee R(\vec{c})$.

We begin with Statement 1. Since $\Xi_M(\mathcal{P})$ is Datalog $\lambda(v)$ contains only one atom. The claim follows since $\mathcal{D}$ contains only predicates in $\mathcal{P}$ and the rules of $\Xi_M(\mathcal{P})$ can only infer facts of one of the three forms. We next show the remaining statements by simultaneous induction on ρ. For the base case, suppose v is the only node in ρ. We distinguish the following four cases:

- If $\lambda(v) \in \mathcal{D}$, then $\mathcal{D} \models \lambda(v)$ holds and the claim is immediate.
- If $\lambda(v) = Q(\vec{b})$, where Q is Horn in $\mathcal{P}$ and $r = (\to Q(\vec{b})) \in \Xi_M(\mathcal{P})$, then $r \in \mathcal{P}$ and the claim follows.
- If $\lambda(v) = \overline{\bot}^Q$, $\mathsf{arity}(Q) = 0$, $\bot \in M$, and $(\to \overline{\bot}^Q) \in \Xi_M(\mathcal{P})$, then the claim reduces to the tautology $\mathcal{P} \models \bot \to Q$.
- If $\lambda(v) = \overline{Q}^Q$ where Q is disjunctive in $\mathcal{P}$ and $\mathsf{arity}(Q) = 0$, then the claim reduces to the tautology $\mathcal{P} \cup \mathcal{D} \models Q \to Q$.

For the inductive step, suppose v has children $v_1, \ldots, v_n$ and, $\lambda(v)$ is a hyperresolvent of $\lambda(v_1), \ldots, \lambda(v_n)$ and a rule $r \in \Xi_M(\mathcal{P})$. We distinguish the following six cases:

- If r contains no disjunctive predicates, then the claim follows since $\mathcal{P} \cup \mathcal{D}$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D}$ entail the same facts over a Horn predicate.

- If $r = \xi_\top \to \overline{R}^R(\vec{y}, \vec{y})$ where R is disjunctive in $\mathcal{P}$, then $\alpha = \overline{R}^R(\vec{b}, \vec{b})$ for some $\vec{b}$, and the claim ($\mathcal{P} \cup \mathcal{D} \models R(\vec{b}) \to R(\vec{b})$) is immediate.

- If $r = R^R(\vec{y}, \vec{y}) \to R(\vec{y})$, then $\lambda(v) = R(\vec{b})$ for some $\vec{b}$. By the inductive hypothesis, $\mathcal{P} \cup \mathcal{D} \models R(\vec{b}) \vee R(\vec{b})$, and the claim is immediate.

- If $r = \xi_\top \wedge \varphi \wedge \bigwedge_{j=1}^m Q_j^R(\vec{t}_j, \vec{y}) \wedge \bigwedge_{i=1}^n \overline{P}_i^R(\vec{s}_i, \vec{y}) \to \overline{Q}^R(\vec{t}, \vec{y})$ where $\xi_\top \wedge \varphi$ is the conjunction of all Horn atoms in r and $r' = \varphi \wedge Q(\vec{t}) \wedge \bigwedge_{j=1}^m Q_j(\vec{t}_j) \to \bigvee_{i=1}^n P_i(\vec{s}_i) \in \mathcal{P}$, then $\lambda(v) = \overline{Q}^R(\vec{b}, \vec{c})$ for some $\vec{b}$ and $\vec{c}$. For some σ, we have $\mathcal{P} \cup \mathcal{D} \models \varphi\sigma$, $\vec{t}\sigma = \vec{b}$ and, for each i, j, $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{P}_i^R(\vec{s}_i\sigma, \vec{c})$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_j^R(\vec{t}_j\sigma, \vec{c})$. Then, by the inductive hypothesis, $\mathcal{P} \cup \mathcal{D} \models P_i(\vec{s}_i\sigma) \to R(\vec{c})$ and $\mathcal{P} \cup \mathcal{D} \models Q_j(\vec{t}_j\sigma) \vee R(\vec{c})$. With r', we obtain $\mathcal{P} \cup \mathcal{D} \models Q(\vec{b}) \to R(\vec{c})$.

- If $r = \varphi \wedge \bigwedge_{j=1}^m Q_j^R(\vec{t}_j, \vec{y}) \wedge \bigwedge_{i=1}^n \overline{P}_i^R(\vec{s}_i, \vec{y}) \to R(\vec{y})$ where φ is the conjunction of all Horn atoms in r and $r' = \varphi \wedge \bigwedge_{j=1}^m Q_j(\vec{t}_j) \to \bigvee_{i=1}^n P_i(\vec{s}_i) \in \mathcal{P}$, then $\lambda(v) = R(\vec{b})$ for some $\vec{b}$. For some σ, we then have $\mathcal{P} \cup \mathcal{D} \models \varphi\sigma$ and, for each i, j, $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{P}_i^R(\vec{s}_i\sigma, \vec{b})$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_j^R(\vec{t}_j\sigma, \vec{b})$. Then, by the inductive hypothesis, $\mathcal{P} \cup \mathcal{D} \models P_i(\vec{s}_i\sigma) \to R(\vec{b})$ and $\mathcal{P} \cup \mathcal{D} \models Q_j(\vec{t}_j\sigma) \vee R(\vec{b})$. With r', we obtain $\mathcal{P} \cup \mathcal{D} \models R(\vec{b})$.

- If $r = \xi_\top \wedge \varphi \wedge \bigwedge_{j=1}^m Q_j^R(\vec{t}_j, \vec{y}) \wedge \bigwedge_{i=1}^n \overline{P}_i^R(\vec{s}_i, \vec{y}) \to P'^R(\vec{s}, \vec{y})$ where $\xi_\top \wedge \varphi$ is the conjunction of all Horn atoms in r and $r' = \varphi \wedge \bigwedge_{j=1}^m Q_j(\vec{t}_j) \to P'(\vec{s}) \vee \bigvee_{i=1}^n P_i(\vec{s}_i)$ in $\mathcal{P}$, then $\lambda(v) = P'^R(\vec{b}, \vec{c})$ for some $\vec{b}$ and $\vec{c}$. For some σ we then have $\mathcal{P} \cup \mathcal{D} \models \varphi\sigma$, $\vec{s}\sigma = \vec{b}$ and, for each i, j, $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models \overline{P}_i^R(\vec{s}_i\sigma, \vec{c})$ and $\Xi_M(\mathcal{P}) \cup \mathcal{D} \models Q_j^R(\vec{t}_j\sigma, \vec{c})$. Then, by the inductive hypothesis, $\mathcal{P} \cup \mathcal{D} \models P_i(\vec{s}_i\sigma) \to R(\vec{c})$ and $\mathcal{P} \cup \mathcal{D} \models Q_j(\vec{t}_j\sigma) \vee R(\vec{c})$. With r', we obtain $\mathcal{P} \cup \mathcal{D} \models P'(\vec{b}) \vee R(\vec{c})$.

This concludes the proofs of Claim (♣), Step 2, and thus Theorem 9. □

5. Rewriting Programs via Unfolding

Unfortunately, in the case of programs that do not satisfy the markability condition we have no algorithmic means to determine whether they can be rewritten into Datalog. Indeed, Bienvenu et al. [22] showed that Datalog rewritability of atomic queries is undecidable w.r.t. ontologies in the Description Logic $\mathcal{ALCF}$. Since each $\mathcal{ALCF}$ ontology admits a Disjunctive Datalog rewriting [23], we can conclude that checking Datalog rewritability (or, equivalently, linearisability) of Disjunctive Datalog programs is an undecidable problem.

In this section, we present a rewriting procedure that combines our results in Section 4 with the work of Gergatsoulis [13] on program transformation for

Procedure 1 Unfold

Input: $\mathcal{P}$: a Disjunctive Datalog program; r: a rule; α: a body atom of r

Output: a Disjunctive Datalog program.

```
 1: $S_0 := \{ (s, \beta) \mid s \in \mathcal{P}, \beta$ a head atom in $s$ unifiable with $\alpha \}$
 2: $i := 0$
 3: repeat
 4:     $S_{i+1} := \emptyset$
 5:     for each $(s, \beta) \in S_i$ do
 6:         $(s', \theta) := \mathsf{ElemUnfold}(r, \alpha, s, \beta)$
 7:         $S_{i+1} := S_{i+1} \cup \{ (s', \beta'\theta) \mid (s, \beta') \in S_i, \beta \neq \beta' \}$
 8:     $i := i + 1$
 9: until $S_i \neq \emptyset$
10: return $(\mathcal{P} \setminus \{r\}) \cup \{ s \mid (s, \beta) \in S_j,$ for $1 \leq j < i \}$
```

Disjunctive Datalog programs. Our procedure iteratively applies the *unfolding* transformation to eliminate rules that preclude markability; it stops when the program becomes markable, in which case it outputs a Datalog program as in Section 4. The procedure is sound: if it succeeds, the output is a Datalog rewriting. It is, however, both incomplete (linearisability cannot be semi-decided just by unfolding) and non-terminating. Nevertheless, our experiments suggest that unfolding can be effective in practice since a number of non-markable Disjunctive Datalog programs obtained from realistic ontologies can be rewritten into Datalog after just a few unfolding steps.

5.1. The Unfolding Transformation

Given a Disjunctive Datalog program $\mathcal{P}$, a rule r in $\mathcal{P}$, and a body atom α of r, Gergatsoulis defines the *unfolding* of r at α in $\mathcal{P}$ as a transformation that replaces r in $\mathcal{P}$ with a set of resolvents of r with other rules in the program, where resolution is applied over α. We next recapitulate this notion.

Definition 10. Let $r = \alpha \wedge \varphi_r \rightarrow \psi_r$ and $s = \varphi_s \rightarrow \beta \vee \psi_s$ be rules such that α is unifiable with β with MGU θ. The *elementary unfolding* $\mathsf{ElemUnfold}(r, \alpha, s, \beta)$ of r at α using s at β is the pair $((\varphi_r \wedge \varphi_s \rightarrow \psi_r \vee \psi_s)\theta, \theta)$. $\diamond$

Elementary unfolding thus amounts to resolving the relevant rules over the given predicates. *Unfolding* is then a transformation where a rule is replaced with a sequence of elementary unfoldings in an equivalence-preserving way.

Definition 11. Let $\mathcal{P}$ be a Disjunctive Datalog program, let $r \in \mathcal{P}$ and let α be a body atom in r; then the *unfolding of r at α in $\mathcal{P}$*, denoted $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$, is the result of applying Procedure 1 to $\mathcal{P}$, r, and α. $\diamond$

As shown by Gergatsoulis, performing all possible elementary unfoldings of r at α using rules in $\mathcal{P}$ does not suffice to eliminate r. Instead, given r and α, Procedure 1 computes a "closure" under elementary unfoldings. Gergatsoulis shows that Procedure 1 terminates (each s' introduced in iteration i occurs in fewer tuples in S_{i+1} than the rule it was obtained from does in S_i), and its output is unique up to variable renaming. The termination argument yields an exponential bound on the size of $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$ in the number of head atoms in $\mathcal{P}$ unifiable with α. Gergatsoulis also shows that unfolding preserves all entailed disjunctions of facts.

The result of Gergatsoulis, however, applies to a *fixed* program and dataset (where the data is incorporated in the program as ground rules). To compute Datalog rewritings via unfolding we need a stronger, data-independent result.

Theorem 12. *Let $\mathcal{P}$ be a Disjunctive Datalog program, r a rule in $\mathcal{P}$, and α an IDB body atom of r. Then, for every dataset $\mathcal{D}$ over the EDB predicates in $\mathcal{P}$ and every disjunction of facts φ, we have $\mathcal{P} \cup \mathcal{D} \models \varphi$ if and only if $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \varphi$.*

Proof. Assume that $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \varphi$. By soundness of resolution, $\mathcal{P} \models \mathsf{Unfold}(\mathcal{P}, r, \alpha)$ and hence $\mathcal{P} \cup \mathcal{D} \models \varphi$.

Assume now that $\mathcal{P} \cup \mathcal{D} \models \varphi$. W.l.o.g., let r be of the form $\bigwedge_{i=1}^{n} \alpha_i \to \psi$, where $n \geq 1$ and $\alpha = \alpha_1$. We show the following claim (♠), which we then exploit to obtain $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \varphi$.

Claim (♠). Let σ be a ground substitution and let $\chi_{\alpha_1}, \dots \chi_{\alpha_n}$ be disjunctions of facts such that $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \alpha_i\sigma \vee \chi_{\alpha_i}$ for every $i \in [1, n]$. Then $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \psi\sigma \vee \chi_{\alpha_1} \vee \dots \vee \chi_{\alpha_n}$.

Let $\rho = (T, \lambda)$ be a derivation, rooted at v, of $\alpha\sigma \vee \chi'_\alpha$ from $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D}$ for some $\chi'_\alpha \subseteq \chi_\alpha$ (ρ exists by completeness of hyperresolution). Let s be the rule used to derive the label of v (i.e., $\alpha\sigma \vee \chi'_\alpha$) from the labels of its children $v_1, \dots, v_m$, and let τ be substitution used in the respective hyperresolution step. Then, $s = \bigwedge_{j=1}^{m} \beta_j \to \alpha'_1 \vee \dots \vee \alpha'_l \vee \psi_{\alpha'}$ with $\lambda(v_j) = \beta_j\tau \vee \chi_{\beta_j}$ for each $j \in [1, m]$, $\alpha\sigma = \alpha'_1\tau = \dots = \alpha'_l\tau$ and $\chi'_\alpha = \psi_{\alpha'}\tau \vee \chi_{\beta_1} \vee \dots \vee \chi_{\beta_m}$. Let r_1 be the rule obtained by elementary unfolding of r at α using s at α'_1, and let r_k ($2 \leq k \leq l$) be the rule obtained by elementary unfolding of r at α using r_{k-1} at α'_k. Then $(\bigwedge_{j=1}^{m} \beta_j\tau) \wedge (\bigwedge_{i=2}^{n} \alpha_i\sigma)$ is a substitution instance of the body of r_l and $\psi_{\alpha'}\tau \vee \psi\sigma$ is the corresponding instance of the head of r_l. Hence, since $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \alpha_i\sigma \vee \chi_{\alpha_i}$ for every $i \in [2, n]$ and, by soundness of hyperresolution, $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \beta_j\tau \vee \chi_{\beta_j}$ for every $j \in [1, m]$, with r_l we obtain $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \psi_{\alpha'}\tau \vee \psi\sigma \vee \chi_{\beta_1} \vee \dots \vee \chi_{\beta_m} \vee \chi_{\alpha_2} \vee \dots \vee \chi_{\alpha_n} = \psi\sigma \vee \chi'_\alpha \vee \chi_{\alpha_2} \vee \dots \vee \chi_{\alpha_n} \subseteq \psi\sigma \vee \chi_\alpha \vee \chi_{\alpha_2} \vee \dots \vee \chi_{\alpha_n} = \psi\sigma \vee \chi_{\alpha_1} \vee \dots \vee \chi_{\alpha_n}$. This concludes the proof of Claim (♠).

We now use Claim (♠) to show $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \varphi$. W.l.o.g., let $\rho = (T, \lambda)$ be a derivation of φ from $\mathcal{P} \cup \mathcal{D}$ (the claim easily adapts if ρ derives some $\varphi' \subseteq \varphi$) and let v be the root of ρ. We proceed by induction on ρ.

For the base case, suppose v is the only node in ρ. Then either $\lambda(v) \in \mathcal{D}$, in which case the claim is immediate, or φ is derived by a rule $s \in \mathcal{P}$ with an empty body. Then the claim follows since $s \neq r$ and hence $s \in \mathsf{Unfold}(\mathcal{P}, r, \alpha)$.

For the inductive step, let $v_1, \dots, v_n$ be the successors of v. By the induction hypothesis, $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \cup \mathcal{D} \models \lambda(v_i)$ for every $i \in [1, n]$. We distinguish two

Procedure 2 Rewrite

Input: $\mathcal{P}$: a Disjunctive Datalog program

Output: a Datalog rewriting of $\mathcal{P}$

```
1: while $\mathcal{P}$ not markable do
2:     compute a minimal pseudo-marking $M'$ of $\mathcal{P}$
3:     select $r \in \mathcal{P}$ with more than one body atom marked by $M'$
4:     select a body atom $\alpha \in r$ marked by $M'$
5:     $\mathcal{P} := \mathsf{DeleteRed}(\mathsf{Unfold}(\mathcal{P}, r, \alpha))$
6: compute a marking $M$ of $\mathcal{P}$
7: return $\Xi_M(\mathcal{P})$
```

cases, depending on the rule $s \in \mathcal{P}$ used to derive $\lambda(v)$ from $\lambda(v_1), \ldots, \lambda(v_n)$. If $s \neq r$, we have $s \in \mathsf{Unfold}(\mathcal{P}, r, \alpha)$, and the claim follows with s. If $s = r$, the claim follows by (♠). □

Theorem 12 implies that $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$ is a rewriting of $\mathcal{P}$ for every r and α.

5.2. The Rewriting Procedure

Our rewriting procedure exploits unfolding to iteratively eliminate rules that violate markability. To identify such culprit rules we exploit a relaxed notion of a marking, in which the first condition in Definition 6 is no longer required.

Definition 13. A *pseudo-marking* of $\mathcal{P}$ is a set M of disjunctive predicates in $\mathcal{P}$ satisfying conditions (2) and (3) in Definition 6. Pseudo-marking M is *minimal* if $\mathcal{P}$ has no pseudo-marking M' such that $M' \subsetneq M$. ◇

The set of all disjunctive predicates in $\mathcal{P}$ is a pseudo-marking of $\mathcal{P}$; thus, every program has a pseudo-marking, and hence also a minimal one. Moreover, if $\mathcal{P}$ is not markable, then for every pseudo-marking M of $\mathcal{P}$ there is a rule in $\mathcal{P}$ with more than one marked body atom (otherwise M would be a marking of $\mathcal{P}$).

In each iteration, our procedure Rewrite computes a minimal pseudo-marking of the current program, finds a rule with more than one marked body atom, and unfolds it on one such atom. The process is repeated (maybe indefinitely) until the program becomes markable. To obtain smaller rewritings and ensure termination in more cases, our procedure employs redundancy elimination. Given a program $\mathcal{P}$, we write $\mathsf{DeleteRed}(\mathcal{P})$ for the program obtained from $\mathcal{P}$ by removing all tautologous rules as well as all rules properly subsumed by other rules in $\mathcal{P}$. Note that for every program $\mathcal{P}$ we have $\mathsf{DeleteRed}(\mathcal{P}) \subseteq \mathcal{P}$ and $\mathsf{DeleteRed}(\mathcal{P}) \models \mathcal{P}$, and thus $\mathcal{P} \equiv \mathsf{DeleteRed}(\mathcal{P})$.

Correctness of Rewrite is established by the following theorem.

Theorem 14. *Let $\mathcal{P}$ be a Disjunctive Datalog program. If* Rewrite *terminates on $\mathcal{P}$ with output $\mathcal{P}'$, then $\mathcal{P}'$ is a Datalog rewriting of $\mathcal{P}$.*

Proof. By Theorem 12, $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$ is a rewriting of $\mathcal{P}$, and so is the logically equivalent program $\mathsf{DeleteRed}(\mathsf{Unfold}(\mathcal{P}, r, \alpha))$. Since the property of being a rewriting is transitive, we conclude that the program obtained in Step 6 of Rewrite is, on the one hand, a rewriting of $\mathcal{P}$ and, on the other hand, a markable program. The claim then follows directly from Theorem 9. □

To conclude this section let us consider as an example the following program $\mathcal{P}_5$, which admits no marking.

$$\begin{aligned}
\mathcal{P}_5 = \{\, & H(x) \rightarrow P(x), && (12)\\
& P(x) \rightarrow M(x) \vee W(x), && (13)\\
& M(x) \rightarrow P(x), && (14)\\
& W(x) \rightarrow P(x), && (15)\\
& M(x) \wedge W(x) \rightarrow \bot \,\} && (16)
\end{aligned}$$

The set $\{M, P, W, \bot\}$ is the only pseudo-marking of $\mathcal{P}_5$. Thus, Rewrite must unfold Rule (16) as it is the only rule containing several marked body atoms. We choose to unfold on atom $M(x)$ and replace (16) with the following rule:

$$P(x) \wedge W(x) \rightarrow W(x) \vee \bot \qquad (17)$$

Rule (17) is a tautology and hence it is eliminated by DeleteRed. Thus, Step 5 in Rewrite amounts to simply deleting Rule (16) from $\mathcal{P}_5$. The resulting program $\mathcal{P}'_5$ is markable (and even linear), with $\{M, P, W\}$ being the only marking. Our procedure then returns the transposition of $\mathcal{P}'_5$ w.r.t. this marking, which by Theorem 14 yields a Datalog rewriting of $\mathcal{P}_5$.

6. Applications to Ontologies

In this section we study the implications of markability on ontology reasoning. After providing a brief overview of Description Logics and ontologies in Section 6.1, we show in Section 6.2 that our notion of markability can be seamlessly adapted to ontologies. Then, we focus on the complexity of reasoning over markable OWL 2 ontologies. In Section 6.3, we study the natural extension of the OWL 2 RL profile based on markability and show that fact entailment in the resulting language is tractable in combined complexity. Subsequently, we consider in Section 6.4 ontologies in the expressive Description Logic $\mathcal{SHI}$ and show that markable $\mathcal{SHI}$ ontologies admit a (possibly exponential size) Datalog rewriting, which implies tractability of fact entailment w.r.t. data complexity (as opposed to co-NP-completeness in the case of unrestricted $\mathcal{SHI}$ ontologies).

6.1. Ontologies and OWL 2

We assume basic familiarity with Description Logics (DLs) and ontology languages and refer the reader to the literature for further details [24, 25]. A DL signature Σ consists of disjoint countable sets of of concept names Σ_C, role names Σ_R, and individuals Σ_I. A role is an element of $\Sigma_R \cup \{R^- \mid R \in \Sigma_R\}$. W.l.o.g., we consider normalised DL axioms as in Table 1. The table also

$A1.$	$\bigsqcap_{i=1}^{n} A_i \sqsubseteq \bigsqcup_{j=1}^{m} C_j$	$\bigwedge_{i=1}^{n} A_i(x) \rightarrow \bigvee_{j=1}^{m} C_j(x)$
$A2.$	$\exists R.A \sqsubseteq B$	$R(x,y) \wedge A(y) \rightarrow B(x)$
$A3.$	$A \sqsubseteq \mathsf{Self}(R)$	$A(x) \rightarrow R(x,x)$
$A4.$	$\mathsf{Self}(R) \sqsubseteq A$	$R(x,x) \rightarrow A(x)$
$A5.$	$R \sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$
$A6.$	$R \sqsubseteq S^{-}$	$R(x,y) \rightarrow S(y,x)$
$A7.$	$R \circ S \sqsubseteq T$	$R(x,z) \wedge S(z,y) \rightarrow T(x,y)$
$A8.$	$A \sqsubseteq \exists R.B$	$A(x) \rightarrow R(x, f_R(x));\ A(x) \rightarrow B(f_R(x))$
$A9.$	$A \sqsubseteq\, \leq m\, R.B$	$A(z) \wedge \bigwedge_{i=1}^{m} R(z,x_i) \wedge B(x_i) \rightarrow \bigvee_{1 \leq i < j \leq m} x_i \approx x_j$
$A10.$	$A \sqsubseteq \{a\}$	$A(x) \rightarrow x \approx a$

Table 1: Normalised DL axioms. $A, B \in \Sigma_C \cup \{\top\}$, $C \in \Sigma_C \cup \{\bot\}$, $R, S, T \in \Sigma_R$, and $a \in \Sigma_I$.

provides the translation π of normalised DL axioms into rules with equality ($\approx$). We will treat equality as an ordinary IDB binary predicate, the meaning of which is axiomatised.

Definition 15. An ontology $\mathcal{O}$ is a finite set of DL axioms. We define $\pi(\mathcal{O})$ as the smallest program containing *(i)* $\pi(\alpha)$ for each axiom α in $\mathcal{O}$, and *(ii)* the standard (Datalog) axiomatisation of equality as a congruence relation over the predicates in $\mathcal{O}$, whenever $\mathcal{O}$ contains an axiom of type A9 or A10.[4] $\diamond$

It can be observed that all axioms in Table 1 correspond to Disjunctive Datalog rules, with the only exception of existentially quantified axioms of type A8, where the corresponding rules contain function symbols.

An ontology consisting of axioms of type A1-A10 in Table 1 is

- $\mathcal{SHIQ}$ if it has no axiom of type A10 and $R = S = T$ in each axiom of type A7;[5]
- $\mathcal{SHI}$ if it is $\mathcal{SHIQ}$ and it contains no axiom of type A9;
- *Horn* if $m = 1$ for each axiom of type A1 and A9, and RL if it is Horn and it has no axiom of type A8.

The logics $\mathcal{SHIQ}$ and $\mathcal{SHI}$ are typically referred to in the literature as expressive DLs due to their high complexity of reasoning: fact entailment for these logics is ExpTime-complete in combined complexity and co-NP-complete w.r.t. data [26]. In contrast, OWL 2 RL is a *lightweight* ontology language, where fact entailment is tractable in combined complexity [7].[6] The favourable

[4]We assume that reflexivity of $\approx$ is axiomatised using the safe rule $\top(x) \rightarrow x \approx x$.

[5]$\mathcal{SHIQ}$ enforces additional restrictions to ensure decidability, which we omit here. We refer the reader to [26] for details.

[6]Datalog reasoning is tractable (as opposed ExpTime-complete) if the maximum number of variables in rules is bounded [4], which is the case for programs stemming from RL ontologies as shown in Table 1.

computational properties of OWL 2 RL have spurred the development of scalable reasoning engines such as GraphDB [8] and RDFox [10].

6.2. Markability for Ontologies

Our notions of weak linearity and markability are formulated for arbitrary first-order programs and hence they can be seamlessly adapted to ontologies.

Definition 16. An ontology $\mathcal{O}$ is *weakly linear* if so is the program $\pi(\mathcal{O})$. Furthermore, we say that a set of predicates M is a *marking* of an ontology $\mathcal{O}$ if it is a marking of $\pi(\mathcal{O})$, and $\mathcal{O}$ is *markable* if it admits a marking. $\diamond$

Example 17. Consider the ontology $\mathcal{O}_1$ and its corresponding program $\pi(\mathcal{O}_1)$:

$$\begin{aligned}\mathcal{O}_1 = \{&\mathsf{Person} \sqsubseteq \mathsf{Man} \sqcup \mathsf{Woman}, \mathsf{Person} \sqsubseteq \exists \mathsf{parent}.\mathsf{Person},\\ &\exists \mathsf{married}.\mathsf{Person} \sqsubseteq \mathsf{Person}, \mathsf{Woman} \sqsubseteq \mathsf{Person}, \mathsf{Man} \sqsubseteq \mathsf{Person}\}\\ \pi(\mathcal{O}_1) = \{&\mathsf{Person}(x) \rightarrow \mathsf{Man}(x) \vee \mathsf{Woman}(x), \mathsf{Person}(x) \rightarrow \mathsf{parent}(x, f(x)),\\ &\mathsf{Person}(x) \rightarrow \mathsf{Person}(f(x)), \mathsf{Person}(y) \wedge \mathsf{married}(x,y) \rightarrow \mathsf{Person}(x),\\ &\mathsf{Woman}(x) \rightarrow \mathsf{Person}(x), \mathsf{Man}(x) \rightarrow \mathsf{Person}(x)\}\end{aligned}$$

Ontology $\mathcal{O}_1$ is markable since $\{\mathsf{Person}, \mathsf{Man}, \mathsf{Woman}\}$ is a marking of $\pi(\mathcal{O}_1)$ $\diamond$.

6.3. Extending OWL 2 RL with Disjunctive Axioms

We next consider a natural extension of OWL 2 RL that allows for an unrestricted use of disjunctive axioms of the form $\bigsqcap_{i=1}^{n} A_i \sqsubseteq \bigsqcup_{j=1}^{m} C_j$. Thus, ontologies in this language correspond to Disjunctive Datalog programs.

Definition 18. An *RL*$^{\sqcup}$ *ontology* is a finite set of DL axioms of type A1–A7 and A9–A10 in Table 1, where $m = 1$ for each axiom of type A9. $\diamond$

Extending OWL 2 RL in this way leads to increased complexity of reasoning since we can now encode standard graph search problems.

Proposition 19. *Fact entailment w.r.t. RL*$^{\sqcup}$ *ontologies is co*-NP*-complete.*

Proof. Membership in co-NP follows from the fact that both the rules in Table 1 and the rules axiomatising equality and $\bot$ contain a bounded number of variables; hence, the corresponding programs can be grounded in polynomial time and entailment in the resulting propositional program can be checked in co-NP. For hardness, it suffices to provide a fairly standard encoding of non-3-colourability (a very similar reduction can be found, e.g., in [27]). Clearly, the following ontology $\mathcal{O}$ can be normalised into an RL$^{\sqcup}$ ontology:

$$\begin{array}{lll} V \sqsubseteq R \sqcup G \sqcup B & B \sqcap \exists \mathsf{edge}.B \sqsubseteq \bot & B \sqcap G \sqsubseteq \bot \\ & G \sqcap \exists \mathsf{edge}.G \sqsubseteq \bot & G \sqcap R \sqsubseteq \bot \\ & R \sqcap \exists \mathsf{edge}.R \sqsubseteq \bot & B \sqcap R \sqsubseteq \bot \end{array}$$

Given an undirected graph $G = (V, E)$, the dataset $\mathcal{D}_G$ contains a fact $V(a)$ for each node $a \in V$ and a fact $\mathsf{edge}(a, b)$ for each edge connecting a and b in E. Then G is non-3-colourable iff $\mathcal{O} \cup \mathcal{D}_G \models \bot$. $\square$

If we restrict RL$^\sqcup$ ontologies $\mathcal{O}$ to be markable, however, we can pick a marking M and exploit the transformation Ξ_M in Definition 8 to compute a Datalog rewriting $\mathcal{P}$ of $\pi(\mathcal{O})$. Tractability of fact entailment for markable RL$^\sqcup$ ontologies is established by the following theorem.

Theorem 20. *Fact entailment w.r.t. markable ontologies expressed in RL$^\sqcup$ is* PTime*-complete in both data and combined complexity.*

Proof. Hardness follows directly from the fact that the problem is already PTime-hard if $\mathcal{O}$ is an RL ontology; thus, we focus on proving membership in PTime. Let M be a marking of $\mathcal{O}$. Let $\pi(\mathcal{O})^e$ be the IDB expansion of $\pi(\mathcal{O})$ and θ the corresponding substitution as defined in Section 2.1, and let $\mathcal{P} = \Xi_M(\pi(\mathcal{O})^e)$.[7] By Theorem 9, $\mathcal{O} \cup \mathcal{D} \models \alpha$ iff $\mathcal{P} \cup \mathcal{D} \models \alpha\theta$. Thus, it suffices to show that the evaluation of $\mathcal{P}$ over $\mathcal{D}$ can be computed in polynomial time in the size of $\mathcal{O}$ and $\mathcal{D}$. First, $\mathcal{P}$ is of size at most quadratic in the size of $\mathcal{O}$, and the arity of a predicate in $\mathcal{P}$ is at most double the arity of a predicate in $\mathcal{O}$. As we can see in Table 1, the rules in $\pi(\mathcal{O})$ contain at most 3 variables; hence, the number of variables in each rule in $\mathcal{P}$ is bounded by 6. Since fact entailment in Datalog is tractable for input programs having a bounded number of variables per rule the claim of the theorem immediately follows. $\square$

6.4. Expressive Ontology Languages

Our Datalog rewriting techniques are not applicable to ontologies containing existentially quantified axioms (i.e., axioms of type A8 in Table 1). Hustadt et al. [23], however, developed an algorithm for transforming a $\mathcal{SHIQ}$ ontology into a Disjunctive Datalog program that preserves entailment of facts over non-transitive roles. This technique was extended by Cuenca Grau et al. [28] to preserve all facts (an alternative translation from an extension of $\mathcal{SHIQ}$ to Disjunctive Datalog that preserves all ground consequences was developed by Rudolph et al. [29]). It follows that every $\mathcal{SHIQ}$ ontology $\mathcal{O}$ admits a Disjunctive Datalog rewriting, the size of which may be exponential in the size of $\mathcal{O}$.

When applied to $\mathcal{SHI}$ ontologies, the algorithm in [23] can be seen as a resolution calculus having binary resolution and factoring as inference rules, which are restricted in a suitable way to ensure termination. The key observation that allows us to transfer our results is captured by the following lemma, which establishes that binary resolution and factoring preserve markability.

Lemma 21. *Let M be a marking of a program $\mathcal{P}$, and let $\mathcal{P}'$ be obtained from $\mathcal{P}$ by applying binary resolution and factoring. Then M is a marking of $\mathcal{P}'$.*

Proof. Note that markability of $\mathcal{P}$ is not affected if $\mathcal{P}$ is extended by factors of rules that are already in $\mathcal{P}$: if M is a marking of r, then M is a marking of

[7]We employ IDB expansion to lift the assumption that datasets contain no IDB facts, which is non-standard for DL ontologies.

every factor of r. As for resolution, it suffices to show that whenever M is a marking of two rules r and r', M is a marking of their resolvent e. W.l.o.g., let

$$\begin{aligned}
r &= \varphi \wedge \bigwedge_{i=1}^{n} \alpha_i \rightarrow \bigvee_{j=1}^{m} \beta_j \vee \alpha \\
r' &= \varphi' \wedge \bigwedge_{i=1}^{n'} \alpha'_i \wedge \alpha' \rightarrow \bigvee_{j=1}^{m'} \beta'_j \\
e &= (\varphi \wedge \varphi' \wedge \bigwedge_{i=1}^{n} \alpha_i \wedge \bigwedge_{i=1}^{n'} \alpha'_i \rightarrow \bigvee_{j=1}^{m} \beta_j \vee \bigvee_{j=1}^{m'} \beta'_j)\sigma
\end{aligned}$$

where φ and φ' contain all Horn atoms in r and r', respectively, and σ is the MGU of α and α'. We distinguish the following cases:

- If α is unmarked, then each α_i is unmarked and each β_j is marked. Thus, e has as many marked body atoms and unmarked head atoms as r'.

- If α is marked and, say, β_1 is unmarked, then each α_i is unmarked, each β_j with $j \geq 2$ is marked, each α'_i is unmarked (since α' is marked), and each β'_j is marked. Thus, e has $\beta_1\sigma$ as the only unmarked head atom and has no marked body atom.

- If every head atom in r is marked (including α), then α' is marked, each α'_i is unmarked, and each β'_j is marked. Consequently, e contains only marked head atoms and as many marked body atoms as r.

Thus, in all three cases M is a marking of e and the claim follows. $\square$

We conclude from Lemma 21 and the results in [23] that markable $\mathcal{SHI}$ ontologies admit a (possibly exponential size) Datalog rewriting.

Theorem 22. *Let $\mathcal{O}$ be a markable $\mathcal{SHI}$ ontology. There exists a Datalog rewriting of $\mathcal{O}$ that can be computed in exponential time in the size of $\mathcal{O}$.*

Proof. Let M be a marking of $\mathcal{O}$. By the results in [23, 28], $\mathcal{O}$ is exponential-time rewritable to a Disjunctive Datalog program $\mathcal{P}$ by means of binary resolution and factoring, where $|\mathcal{P}|$ is exponentially bounded in $|\mathcal{O}|$. By Lemma 21, M is a marking of $\mathcal{P}$. Then $\Xi_M(\mathcal{P}^e)$ is a Datalog rewriting of $\mathcal{P}$, and hence of $\mathcal{O}$. The claim follows since $\Xi_M(\mathcal{P}^e)$ is polynomial-time computable in $|\mathcal{P}|$. $\square$

We can now establish tight bounds on the complexity of reasoning over markable $\mathcal{SHI}$ ontologies.

Theorem 23. *Fact entailment w.r.t. markable ontologies expressed in $\mathcal{SHI}$ is* PTime-*complete in data and* ExpTime-*complete in combined complexity.*

PROOF. The EXPTIME upper bound is immediate since fact entailment over unrestricted $\mathcal{SHI}$ ontologies is known to be EXPTIME-complete. Furthermore, the PTIME upper bound for data complexity follows directly from Theorem 22.

The lower bounds follow from existing results for Horn-$\mathcal{ALC}$, where we say that a $\mathcal{SHI}$ ontology is Horn-$\mathcal{ALC}$ if it is Horn and contains no axiom of type A7. Indeed, fact entailment for Horn-$\mathcal{ALC}$ is EXPTIME-hard [30] and PTIME-hard in data; furthermore, every Horn-$\mathcal{ALC}$ ontology is trivially markable as it corresponds to a Horn program. $\square$

7. Conjunctive Queries

We have so far been interested in computing Datalog rewritings that preserve the entailment of facts (or, equivalently, the answers to all atomic queries). We now shift our attention to Datalog rewritings that preserve the answers to unrestricted CQs.

In this setting, it is no longer possible to obtain query-independent rewritings. Lutz and Wolter [14] showed that for *any* Disjunctive Datalog program $\mathcal{P}$ containing at least one disjunctive rule there exists a conjunctive query q such that answering the (fixed) query q w.r.t. the (fixed) program $\mathcal{P}$ and an input dataset is co-NP-hard. Under standard complexity-theoretic assumptions, this implies that there cannot exist a Datalog rewriting of $\mathcal{P}$ that preserves the answers to all CQs (and, in particular, those of q).

If we impose restrictions on *both* program $\mathcal{P}$ and query q, however, it is still possible to identify situations where it is feasible to compute a Datalog rewriting. In this section, we investigate classes of queries and Disjunctive Datalog programs that admit Datalog rewritings, and we discuss the implications of these results on ontology reasoning.

7.1. Characterising Datalog Rewritability for Conjunctive Queries

In Section 3, we established a characterisation of Datalog rewritability based on linearity: a Disjunctive Datalog program $\mathcal{P}$ is rewritable into a Datalog program $\mathcal{P}_1$ iff it is rewritable into a linear Disjunctive Datalog program $\mathcal{P}_2$. In this case, the requirement imposed on such $\mathcal{P}_1$ and $\mathcal{P}_2$ is that they preserve the answers to all atomic queries for every dataset over the signature of $\mathcal{P}$.

We next show that this characterisation can be seamlessly lifted to CQs. For this, it suffices to observe that answering a CQ $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$ w.r.t. a Disjunctive Datalog program $\mathcal{P}$ reduces to answering the atomic query $Q(\vec{x})$ w.r.t. the program $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \to Q(\vec{x})\}$, for Q a fresh predicate.

Theorem 24. *A CQ q is Datalog rewritable w.r.t. a Disjunctive Datalog program $\mathcal{P}$ iff q is rewritable w.r.t. $\mathcal{P}$ into linear Disjunctive Datalog.*

PROOF. Let $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$, and let Q be a fresh predicate of the same arity as q. Then, for every $\mathcal{D}$ and $\vec{a}$, $\mathcal{P} \cup \mathcal{D} \models q(\vec{a})$ iff $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \to Q(\vec{x})\} \cup \mathcal{D} \models Q(\vec{a})$. It suffices to show that $Q(\vec{x})$ is Datalog rewritable w.r.t. $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \to Q(x)\}$ iff $Q(\vec{x})$ is rewritable w.r.t. $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \to Q(x)\}$ into linear Disjunctive Datalog.

This, in turn, follows by Theorem 3 and Proposition 2: if $\mathcal{P}'$ is a Datalog (or linear Disjunctive Datalog) rewriting of $Q(\vec{x})$ w.r.t. $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(x)\}$, then $\Xi(\mathcal{P}')$ is a linear Disjunctive Datalog (resp., Datalog) rewriting of $Q(\vec{x})$ w.r.t. $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(x)\}$. □

7.2. Exploiting the Markability Condition

A Datalog rewriting of a fixed atomic query $Q(\vec{x})$ w.r.t. a markable program $\mathcal{P}$ must only preserve the answers to Q (rather than the answers to all atomic queries). Thus, the rewriting $\Xi_M(\mathcal{P})$ can be optimised by deleting all rules involving auxiliary predicates X^R and $\overline{X}^R$ for $R \neq Q$. In particular, if Q is a Horn predicate in $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(\vec{x})\}$, the optimised rewriting does not need to contain any auxiliary predicates.

Definition 25. Let $\mathcal{P}$ be a Disjunctive Datalog program, let M be a marking of $\mathcal{P}$, and let S be a set of predicates. The *M-transposition of $\mathcal{P}$ w.r.t. S* is the program $\Xi^S_M(\mathcal{P})$ obtained from $\Xi_M(\mathcal{P})$ by removing all rules involving a predicate X^R or $\overline{X}^R$ for $R \notin S$. ◇

The transposition $\Xi^S_M(\mathcal{P})$ is linear in the size of $\mathcal{P}$ for a fixed S. It is easily seen that $\Xi^S_M(\mathcal{P})$ is a rewriting of each atomic query over predicates in S.

Theorem 26. *Let $\mathcal{P}$ be a Disjunctive Datalog program, let M be a marking of $\mathcal{P}$, and let S be a set of predicates. Then, $\Xi^S_M(\mathcal{P})$ is a Datalog rewriting of all atomic queries over S w.r.t. $\mathcal{P}$.*

Proof. The claim is shown analogously to Theorem 9 with the additional observation that Step 1 of the proof only needs rules involving auxiliary predicates of the form $\overline{Q}^P$ and Q^P to show facts about a predicate P. □

Based on these observations, we can exploit our results on markability to identify a class of Disjunctive Datalog programs and CQs admitting a Datalog rewriting, and for which CQ entailment is tractable in data complexity.

Theorem 27. *Let $\mathcal{P}$ be a Disjunctive Datalog program and let $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$ be a CQ. Furthermore, assume that there exists some marking M of $\mathcal{P}$ that marks at most one atom of q, and let Q be a fresh predicate of arity $|\vec{x}|$. Then the following properties hold:*

1. *$\Xi^{\{Q\}}_M(\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(\vec{x})\})$ is a Datalog rewriting of q w.r.t. $\mathcal{P}$.*
2. *For every tuple of constants $\vec{a}$ with $|\vec{a}| = |\vec{x}|$, answering the (fixed) Boolean CQ $q(\vec{a})$ w.r.t. (fixed) $\mathcal{P}$ and arbitrary data is a tractable problem.*

Proof. Statement 2 in the theorem follows directly from the first one, which we show next. Clearly, $\mathcal{P} \cup \mathcal{D} \models q(\vec{a})$ if and only if $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(\vec{x})\} \cup \mathcal{D} \models Q(\vec{a})$ for every $\mathcal{D}$ and $\vec{a}$, and hence $\Xi^{\{Q\}}_M(\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(\vec{x})\})$ is a Datalog rewriting of q w.r.t. $\mathcal{P}$ provided M is a marking of $\mathcal{P} \cup \{\varphi(\vec{x}, \vec{y}) \rightarrow Q(\vec{x})\}$. This holds since M is a marking of $\mathcal{P}$, φ has at most one marked atom, and Q does not occur in $\mathcal{P}$. □

Since $\mathrm{RL}^{\sqcup}$ ontologies correspond to Disjunctive Datalog programs, the result in Theorem 27 also applies to markable $\mathrm{RL}^{\sqcup}$ ontologies. Note also that the theorem is always applicable whenever the input query has at most one disjunctive atom. As we will see in our evaluation, in typical non-Horn ontologies more than 70% of predicates are Horn; hence, it is reasonable to expect that the theorem will be applicable to many queries in practice.

The following example illustrates the rewriting of a conjunctive query w.r.t. an $\mathrm{RL}^{\sqcup}$ ontology.

Example 28. Consider the following $\mathrm{RL}^{\sqcup}$ ontology $\mathcal{O}$ and query q:[8]

$$\begin{aligned} \mathcal{O} &= \{A \sqsubseteq B' \sqcup C',\ B \sqsubseteq B',\ C \sqsubseteq C'\} \\ q(x) &= \exists y z_1 z_2.\, R(x,y) \wedge R(y,z_1) \wedge R(y,z_2) \wedge B'(z_1) \wedge C'(z_2) \end{aligned}$$

The program $\pi(\mathcal{O})$ corresponding to $\mathcal{O}$ is as follows:

$$A(x) \to B'(x) \vee C'(x) \tag{18}$$
$$B(x) \to B'(x) \tag{19}$$
$$C(x) \to C'(x) \tag{20}$$

We can check that the Datalog program $\mathcal{P} = \{B(x) \to B'(x),\ C(x) \to C'(x)\}$ is a rewriting of $\mathcal{O}$. This program, however, is not a rewriting of q, as witnessed by the following dataset $\mathcal{D}$ for which $\mathcal{O} \cup \mathcal{D} \models q(a)$:

$$\{R(a,b_1), R(a,b_2), R(b_1,c_1), R(b_1,c_2), R(b_2,c_2), R(b_2,c_3), B(c_1), A(c_2), C(c_3)\}$$

Clearly, $M = \{B'\}$ is a marking of $\mathcal{O}$, and q contains one marked atom. Moreover, $M' = \{B', Q\}$ is a marking of $\pi(\mathcal{O}) \cup \{r_q\}$, where

$$r_q = R(x,y) \wedge R(y,z_1) \wedge R(y,z_2) \wedge B'(z_1) \wedge C'(z_2) \to Q(x)$$

and $\mathcal{P}' = \Xi^{\{Q\}}_{M'}(\pi(\mathcal{O}) \cup \{r_q\})$ contains the following rules:

$$A(x) \wedge \overline{B'}^{Q}(x,y) \to C'^{Q}(x,y) \tag{18'}$$
$$B(x) \wedge \overline{B'}^{Q}(x,y) \to Q(y) \tag{19'}$$
$$\top(y) \wedge C(x) \to C'^{Q}(x,y) \tag{20'}$$
$$\overline{Q}^{Q}(x,u) \wedge R(x,y) \wedge R(y,z_1) \wedge R(y,z_2) \wedge C'^{Q}(z_2,u) \to \overline{B'}^{Q}(z_1,u) \tag{21}$$
$$\top(x) \to \overline{Q}^{Q}(x,x) \tag{22}$$

Then, $\mathcal{P}'$ is a rewriting of q; Fig. 2 shows a derivation of $Q(a)$ from $\mathcal{P}' \cup \mathcal{D}$. $\diamond$

To conclude this section, observe that Theorem 27 only transfers to $\mathcal{SHI}$ (or potentially $\mathcal{SHIQ}$) ontologies if the CQ q corresponds to a normalised $\mathcal{SHIQ}$ axiom. This is because the reduction to Disjunctive Datalog in [23, 28] is only complete for inputs equivalent to $\mathcal{SHIQ}$ ontologies.

[8]This example is based on a personal communication with Carsten Lutz.

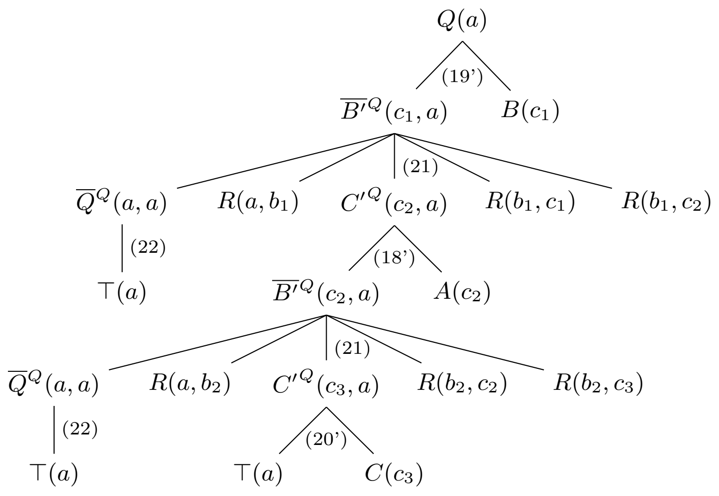


Figure 2: Derivation of $Q(a)$ from $\mathcal{P}' \cup \mathcal{D}$ in Example 28.

8. Related Work

The computational complexity and expressive power of Disjunctive and plain Datalog are well-understood, and we refer the interested reader to [4] for an excellent survey. Disjunctive Datalog with negation as failure has also been extensively studied in the Logic Programming literature [3, 31].

The specification of fragments of (Disjunctive) Datalog with more favourable computational properties has also received significant attention. The class of *head-cycle free* Disjunctive Datalog programs with negation as failure was studied by Ben-Eliyahu-Zohary et al. [31, 32], who showed that certain reasoning problems are tractable for such programs (fact entailment, however, remains intractable in data complexity). In particular, head-cycle free programs are amenable to a program transformation technique known as *shifting* [33–35] that bears some resemblance to program transposition. Semantic characterisations of first-order rewritability in the context of non-monotonic programs were studied in [36]. Furthermore, there is a large body of work on Datalog and first-order rewritability of Horn Description Logics and related languages [37–45]. Finally, linearity has been extensively studied in the context of plain Datalog and it is known to limit the effect of recursion and lead to reduced complexity of reasoning, namely NLogSpace vs PTime in the case of data complexity and PSpace vs ExpTime for combined complexity [11].

Gottlob et al. [46] investigated the complexity of reasoning over disjunctive tuple-generating dependencies (TGDs), which extend Disjunctive Datalog by allowing existentially quantified variables in the head of rules. In particular, they showed tractability (in data complexity) of fact entailment for a class of disjunctive TGDs with single-atom bodies. This class of rules is incomparable to weakly-linear and markable Disjunctive Datalog as it allows existential quantification (and hence function symbols) in the head of rules. Artale et al. [47] showed tractability of fact entailment w.r.t. data for DLs of the DL-Lite$_{\mathsf{bool}}$ family. This result is strongly related to that in [46] since many variants of DL-Lite$_{\mathsf{bool}}$ can be normalised as disjunctive TGDs with singleton bodies. Finally, combined complexity of CQ answering for different classes of disjunctive TGDs was studied by Bourhis et al. [48].

Lutz and Wolter [14] investigated non-uniform data complexity of CQ answering w.r.t. extensions of $\mathcal{ALC}$, and related CQ answering to constraint satisfaction problems. This connection was explored by Bienvenu et al. [22], who showed NExpTime-completeness of first-order and Datalog rewritability of atomic queries for $\mathcal{SHI}$.

A goal-directed resolution procedure for computing first-order and Datalog rewritings of Disjunctive Datalog programs was proposed in [28] and further refined in [2]. Termination of the procedure in [28] is guaranteed for DL-Lite$_{\mathsf{bool}}$ logics, and it was extended to a restricted class of $\mathcal{SHI}$ ontologies in [2]. Both classes of of ontologies are incomparable to WL or markable ontologies. The procedures in [2, 28] do not run in polynomial time or compute polynomial-size rewritings. The procedure in [28] is used in [49] to show first-order/Datalog rewritability of two fragments of the DL $\mathcal{ELU}$. Notably, both fragments yield linear programs and hence are subsumed by those studied in this paper.

Transposition bears a superficial resemblance to the Magic Sets method [50] for Datalog, which has been extended to apply also to Disjunctive Datalog programs [51, 52]. Note, however, that the goal of Magic Sets is not to eliminate the need for disjunctive reasoning (the Magic Set transformation of a Disjunctive Datalog program will still contain disjunctive rules), but rather to restrict bottom-up computation in the presence of a query so as to only compute facts that are relevant to the query; in particular, Magic Sets are inherently query-dependent while our technique is essentially query-independent. Transposition and Magic Sets are thus largely orthogonal.

The idea of merging two atoms of smaller arity into one atom of larger arity has been exploited by Faber and Woltran [53], albeit for a different purpose and with a different intended semantics for the merged atoms.

Finally, our unfolding-based rewriting procedure is motivated by the work of Afrati et al. [54] on linearisation of plain Datalog programs by means of program transformation techniques [12, 13, 55].

9. Practical Considerations

In this section we discuss various optimisations that were instrumental in applying our rewriting techniques to ontological reasoning over large datasets.

9.1. Optimising Transposition

Our transposition transformation introduces body atoms over the unary predicate $\top$ to ensure safety of the resulting rules (recall Definitions 1 and 8). This, however, can lead to performance issues.

Consider again the example program $\mathcal{P}_1$ from Section 3.1. The initialisation rule $\top(x) \to \overline{B}^B(x,x)$ in $\Xi(\mathcal{P}_1)$ derives a fact $\overline{B}^B(a,a)$ for each constant a in the input dataset $\mathcal{D}$, which can then trigger the derivation of new facts. An auxiliary fact $\overline{B}^B(a,a)$, however, is only relevant to the rewriting if $B(a)$ follows from $\mathcal{P}_1 \cup \mathcal{D}$, and hence initialising $\overline{B}^B$ with all constants may lead to unnecessarily large materialisations. Although the facts derivable from $\mathcal{P}_1 \cup \mathcal{D}$ are unknown in advance, we can overapproximate them. For this, we introduce fresh predicates B_u and G_u representing the overapproximation of the disjunctive predicates B and G and construct a program $\mathcal{P}'_1$ by replacing every disjunctive predicate with its corresponding fresh predicate and splitting all disjunctive rules into different Datalog rules as given next:

$$\begin{array}{ll} C(x) \to B_u(x) & G_u(y) \wedge E(x,y) \to B_u(x) \\ C(x) \to G_u(x) & B_u(y) \wedge E(x,y) \to G_u(x) \end{array}$$

Finally, we add to $\mathcal{P}'_1$ all rules in the transposition $\Xi(\mathcal{P}_1)$ while replacing $\top$ in the initialisation rules (4) with the freshly introduced predicates as follows:

$$X_u(x) \to \overline{X}^X(x,x) \quad \text{for each } X \in \{B,G\}$$

In this way, the predicates $\overline{X}^X$ are initialised with the extension of X_u. Consider now a fact $\overline{X}^X(a,a)$. If it is not included in the materialisation of $\mathcal{P}'_1 \cup \mathcal{D}$ then neither is $X_u(a)$; but then, since X_u overapproximates X, we also have that $X(a)$ does not follow from $\mathcal{P}_1 \cup \mathcal{D}$, which implies that $\overline{X}^X(a,a)$ is irrelevant.

We next capture these ideas formally by defining the optimised transposition Ξ^u_M as an extension of Ξ_M. The definition can be straightforwardly adapted to the transformations Ξ and Ξ^S_M in Definitions 1 and 25, respectively.

Definition 29. Let $\mathcal{P}$ be a Disjunctive Datalog program and let M be a marking of $\mathcal{P}$. For each disjunctive predicate Q in $\mathcal{P}$, let Q_u be fresh and of the same arity as Q. Then, $\Xi^u_M(\mathcal{P})$ is the smallest program with all rules given next:

1. $(\bigwedge_{i=1}^n \alpha_i \to \beta_j)\theta$ for each rule $\bigwedge_{i=1}^n \alpha_i \to \bigvee_{j=1}^m \beta_j$ in $\mathcal{P}$ and $j \in [1,m]$, where θ maps every disjunctive predicate Q to Q_u;

2. every rule in $\mathcal{P}$ with no disjunctive predicates;

3. every rule in $\Xi_M(\mathcal{P})$ of type 3 or 5 in Definition 8;

4. $R_u(\vec{y}) \to \overline{R}^R(\vec{y},\vec{y})$ for each rule $\xi_\top \to \overline{R}^R(\vec{y},\vec{y})$ in $\Xi_M(\mathcal{P})$;

5. $Q_u(\vec{t}) \wedge R_u(\vec{y}) \wedge \psi \to \overline{Q}^R(\vec{t},\vec{y})$ for each $\xi_\top \wedge \psi \to \overline{Q}^R(\vec{t},\vec{y})$ in $\Xi_M(\mathcal{P})$; and

6. $P_u(\vec{s}) \wedge R_u(\vec{y}) \wedge \psi \to P^R(\vec{s},\vec{y})$ for each $\xi_\top \wedge \psi \to P^R(\vec{s},\vec{y})$ in $\Xi_M(\mathcal{P})$. $\diamond$

We can show that this optimised transformation can also be exploited to polynomially rewrite markable programs into Datalog.

Theorem 30. *Let $\mathcal{P}$ be a Disjunctive Datalog program and let M be a marking of $\mathcal{P}$. Then $\Xi^u_M(\mathcal{P})$ is a polynomial-size Datalog rewriting of $\mathcal{P}$.*

PROOF (SKETCH). Let $\mathcal{D}$ be a dataset. First, we note that the new predicates in $\Xi_M^u(\mathcal{P})$ overapproximate the disjunctive predicates in $\mathcal{P}$ in the following sense. Claim ($\clubsuit$). For every disjunction of facts φ derivable by hyperresolution from $\mathcal{P} \cup \mathcal{D}$ and every disjunctive atom $Q(\vec{a}) \in \varphi$, we have $\Xi_M^u(\mathcal{P}) \cup \mathcal{D} \models Q_u(\vec{a})$.

The claim follows by a simple induction on a derivation of φ from $\mathcal{P} \cup \mathcal{D}$. Using this claim, the proof of Theorem 9 can be easily adapted with the following observations.

1. Rules in $\Xi_M(\mathcal{P})$ with head atoms of the form $Q^R(\vec{s}, \vec{y})$ or $\overline{Q}^R(\vec{s}, \vec{y})$ can only participate in derivations of facts about the predicate R (among all predicates in $\mathcal{P}$). Thus, by Claim ($\clubsuit$), the body of every such rule can be extended with $R_u(\vec{y})$ without affecting the rule's consequences over the predicates in $\mathcal{P}$.

2. Whenever a rule $r \in \Xi_M(\mathcal{P})$ with a head of the form $Q^R(\vec{s}, \vec{y})$ or $\overline{Q}^R(\vec{s}, \vec{y})$ is used in the proof of Theorem 9 with some substitution σ, we have $\mathcal{P} \cup \mathcal{D} \vdash \varphi$ where $Q(\vec{s}\sigma) \in \varphi$. By Claim ($\clubsuit$), we have $\Xi_M^u \cup \mathcal{D} \models Q_u(\vec{s}\sigma)$, and hence the body of r can be extended with $Q_u(\vec{s})$ without affecting the rule's consequences over the predicates in $\mathcal{P}$. $\Box$

Even with this optimisation, the presence of transposed rules that rely on predicates P_u for safety may still lead to performance issues. A common source of such problematic rules are constraints of the form $\varphi \wedge A(x) \to \bot$ where φ is a conjunction of Horn atoms mentioning variable x and A is marked. Such rules are rewritten to $\varphi \wedge \overline{\bot}^P(\vec{y}) \to \overline{A}^P(x, \vec{y})$ for every predicate P, where the extension of $\overline{\bot}^P$ is determined by the rule $P_u(\vec{x}) \to \overline{\bot}^P(\vec{x})$. Thus, the extension of $\overline{A}^P$ in $\Xi_M^u(\mathcal{P}) \cup \mathcal{D}$ is populated by the Cartesian product of the extension of φ and the extension of P_u. Hence, even if φ and P_u are both linearly bounded in the size of $\mathcal{D}$, the extension of $\overline{A}^P$ will be quadratic in $\mathcal{D}$.

This can often be remedied by unfolding the problematic rules. For instance, let $r = E(x) \wedge A(x) \to \bot \in \mathcal{P}$ where E is Horn in $\mathcal{P}$, and let $B(x) \wedge C(x) \to A(x)$ and $D(x) \to A(x) \vee F(x)$, with B, C, D, F all disjunctive in $\mathcal{P}$, be the only rules in $\mathcal{P}$ where predicate A occurs in the head. Then $\mathsf{Unfold}(\mathcal{P}, r, A(x))$ replaces r in $\mathcal{P}$ with the rules $E(x) \wedge B(x) \wedge C(x) \to \bot$ and $E(x) \wedge D(x) \to \bot \vee F(x)$. Unlike r, the new rules each contain at least two disjunctive atoms besides $\bot$, and hence their rewritings contain at least one disjunctive body atom that is not $\overline{\bot}^P(\vec{y})$, typically leading to smaller materialisations.

We can realise this idea using Procedure 3, which unfolds rules of the form $\varphi \wedge \alpha \to \bot$ where α is marked by M and φ is a conjunction of Horn atoms. Note that the procedure terminates for every $\mathcal{P}$ and M since every iteration of the main loop that modifies $\mathcal{P}$ strictly reduces the number of problematic rules. Procedure $\mathsf{RewriteConstraints}$ can then be incorporated into Procedure $\mathsf{Rewrite}$ from Section 5 by replacing $\Xi_M(\mathcal{P})$ with $\Xi_M^u(\mathsf{RewriteConstraints}(\mathcal{P}, M))$ in Step 7 of the procedure. This modification preserves correctness of $\mathsf{Rewrite}$ since *(i)* the program computed by $\mathsf{RewriteConstraints}$ for a program $\mathcal{P}$ and any M is a rewriting of $\mathcal{P}$ (Theorem 12), and *(ii)* every marking M of $\mathcal{P}$ is a marking of $\mathsf{RewriteConstraints}(\mathcal{P}, M)$, as established by Lemma 21.

Procedure 3 RewriteConstraints

Input: $\mathcal{P}$: a Disjunctive Datalog program; M: a marking of $\mathcal{P}$

Output: a rewriting of $\mathcal{P}$

```
1: for each $r = \varphi \wedge \alpha \to \bot$ in $\mathcal{P}$ with $\alpha$ marked and $\varphi$ Horn do
2:     $\mathcal{P}' := \mathsf{Unfold}(\mathcal{P}, r, \alpha)$
3:     if $\mathcal{P}' \setminus \mathcal{P}$ has no rule $\varphi' \wedge \alpha' \to \bot$ with $\alpha'$ marked and $\varphi'$ Horn then
4:         $\mathcal{P} := \mathcal{P}'$
5: return $\mathcal{P}$
```

9.2. Guiding the Unfolding

The success of our rewriting procedure Rewrite described in Section 5 largely depends on the choice of rule and atom to unfold in Steps 3 and 4. In our implementation we have refined the rule selection strategy given in Steps 2 and 3. Rather than computing a pseudo-marking of $\mathcal{P}$, we exploit instead the data structures used for markability checking; specifically, we use the notion of *implication graph* introduced by Aspvall et al. [56] for solving 2-SAT.

Definition 31. Let $\mathcal{N}$ be a 2-SAT instance. Given a proposition literal l, we write $\bar{l}$ for the negation of l, i.e., $\bar{l} = \neg X$ if $l = X$ and $\bar{l} = X$ if $l = \neg X$. The *implication graph* of $\mathcal{N}$ is the smallest digraph $G_{\mathcal{N}}$ containing nodes l and $\bar{l}$ for each literal l in $\mathcal{N}$, and edges $(\bar{l}_1, l_2)$ and $(\bar{l}_2, l_1)$ for each clause $l_1 \vee l_2$ in $\mathcal{N}$.

A *clash* in $G_{\mathcal{N}}$ is a directed cycle involving some literal l and its negation $\bar{l}$. A literal is *clashing* if it is involved in a clash. ◇

Let $\mathcal{P}$ be a program and $\mathcal{N}$ the 2-SAT instance encoding its markability. Then $\mathcal{P}$ is markable iff $\mathcal{N}$ is satisfiable iff $G_{\mathcal{N}}$ contains no clash (the latter equivalence due to Aspvall et al. [56]). The following proposition establishes that, in order to obtain a markable program, it suffices to remove from $\mathcal{P}$ all rules that contain at least two clashing body atoms. Thus, in our implementation of Rewrite we always select one such rule in Step 3.

Proposition 32. *Let $\mathcal{P}$ be a Disjunctive Datalog program and let $\mathcal{N}$ be the 2-SAT instance encoding markability of $\mathcal{P}$. Then $\mathcal{P}$ is markable if and only if it contains no rule that has two distinct body atoms $P(\vec{s})$, $Q(\vec{t})$ such that X_P, X_Q are clashing in $G_{\mathcal{N}}$.*

Proof. Assume that $\mathcal{P}$ is markable. Then $\mathcal{N}$ is satisfiable, and hence no literal is clashing in $G_{\mathcal{N}}$, which trivially implies the claim. Assume now that $\mathcal{P}$ is not markable. Then, $\mathcal{P}$ contains a rule r with two distinct disjunctive body atoms $P(\vec{s})$, $Q(\vec{t})$ such that $(X_P, \neg X_Q)$ or $(X_Q, \neg X_P)$ is part of a clash in $G_{\mathcal{N}}$; note that edges coming only from clauses of the form (2) and (3) in $\mathcal{N}$ cannot lead to a clash, which needs to be present since $\mathcal{P}$ is not markable. Let l and $\bar{l}$ be the clashing literals and let, w.l.o.g., $\pi_1 = l, \ldots, X_P, \neg X_Q, \ldots, \bar{l}$ be a directed path from l to $\bar{l}$ and π_2 be a directed path from $\bar{l}$ to l. An easy induction on n reveals that for every path $l_1, \ldots, l_n$ in $G_{\mathcal{N}}$, the sequence $\bar{l}_n, \ldots, \bar{l}_1$ also constitutes a path in $G_{\mathcal{N}}$. From this we can conclude that $\pi_1' = l, \ldots, X_Q, \neg X_P, \ldots, \bar{l}$ is a path from l to $\bar{l}$. We next argue that both X_P and X_Q are clashing. For X_P, the concatenation of the subpath $X_P, \neg X_Q, \ldots, \bar{l}$ of π_1, the path π_2, and the subpath $l, \ldots, X_Q, \neg X_P$ of π_1' is a path from X_P to $\neg X_P$. Moreover, the concatenation of $\neg X_P, \ldots, \bar{l}$ from π_1', π_2 and $l, \ldots, X_P$ from π_1 is a path from $\neg X_P$ to X_P. Analogously, the concatenation $\pi_1' \pi_2 \pi_1 \pi_2$ is a directed cycle between X_Q and $\neg X_Q$, as required. $\square$

The choice of atom to unfold on in Step 4 of Rewrite is determined heuristically. We choose an atom with minimal "depth", as defined next.

Definition 33. Let $G_{\mathcal{P}}$ be the dependency graph of a program $\mathcal{P}$. Given predicates P, Q in $\mathcal{P}$, let $P \sim Q$ hold iff P and Q belong to the same strongly connected component of $G_{\mathcal{P}}$. The *depth of an atom $Q(\vec{t})$ in $\mathcal{P}$* is the depth of the strongly connected component of Q in the quotient graph $G_{\mathcal{P}}/_{\sim}$.[9] $\diamond$

This heuristic aims at avoiding unfolding cycles where predicates are repeatedly replaced with predicates from the same strongly connected component.

10. Proof of Concept Evaluation

We have implemented weak linearity and markability checkers as well as the unfolding-based rewriting procedure Rewrite in Section 5.2. Additionally, we have implemented the transposition transformation for markable Disjunctive Datalog programs described in Sections 4 and 9.1.

We have conducted two kinds of experiments, the results of which are described in Sections 10.1 and 10.2.

- *Rewritability experiments*, where we have evaluated whether the non-Horn ontologies available in well-known ontology repositories can be rewritten into Datalog using our techniques. To this end, we have first exploited the KAON2 reasoner [57] to transform ontologies into Disjunctive Datalog and then applied our rewriting algorithms to the resulting programs.

- *Query answering experiments*, where we have evaluated the potential benefits of our approach for optimising query answering over Disjunctive Datalog programs obtained from ontologies. We considered two non-Horn ontologies that, on the one hand, come with a large-scale dataset and, on the other hand, correspond to a markable Disjunctive Datalog program.

All experiments were conducted on a machine equipped with two Intel Xeon E5-2670 processors and 256GB RAM. Our prototype implementation as well as all test ontologies and datasets used in the evaluation are available online.[10]

[9]Note that the quotient graph is a DAG and hence the notion of depth is standard.
[10]https://krr-nas.cs.ox.ac.uk/2015/AIJ/Rewriting/.

10.1. Rewritability Experiments

We collected non-Horn ontologies from BioPortal,[11] the Protégé library,[12] the corpus of Gardiner et al. [58], the EBI RDF Platform,[13] and the repository of the ISKP Group at the University of Thessaloniki.[14] In total, we gathered 128 ontologies from these sources.

The version of KAON2 available to us is restricted to the $\mathcal{SHIN}$ fragment of OWL DL. Hence, we modified some of the test ontologies in a minimal possible way so that they are accepted by the KAON2 parser. In addition to ineffectual changes (e.g., removing all annotations) we made the following modifications that may have an impact on rewritability:

1. We turned data properties into object properties and replaced all datatype related axioms with their corresponding axioms for object properties.
2. We approximated nominals (i.e., axioms A10 in Table 1 from Section 6.1) by replacing them with fresh classes in the usual way [59], and approximated self restrictions and qualified number restrictions using existential and universal restrictions.
3. We removed reflexive and irreflexive properties as well as property chain axioms from the ontology, and then added them back as Datalog rules to the Disjunctive Datalog program computed by KAON2.

In total, we approximated 61 out of our 128 test ontologies in this way. Note that the changes in Point 1 may only facilitate rewritability if there exist semantic dependencies between the datatypes used in the ontology. Furthermore, the approximation in Point 3 implies that the mentioned property axioms are not taken into account during the resolution stage of KAON2, but they are considered later on for markability checking; although this is a sound but incomplete approximation, it a most faithful one given the restrictions of KAON2.

Out of the 128 ontologies we considered, 16 ontologies were already $\mathrm{RL}^{\sqcup}$ and could be directly normalised as Disjunctive Datalog programs. From the remaining 112 ontologies, KAON2 succeeded in computing Disjunctive Datalog programs in 99 cases. Out of the total of 115 Disjunctive Datalog programs, we found that 40 were markable (with 36 of them already WL, and where 9 of them had been modified for KAON2). Furthermore, our unfolding-based procedure Rewrite succeeded on 6 additional programs when given a time limit of 60 seconds (5 of which were modified for KAON2); these programs required no more than 34 unfolding steps, and the programs after unfolding were no more than 50% larger than the original ones (with one exception, where the unfolded program was 6 times larger than the original program while taking only 3 unfolding steps to compute). On average, 78% of the predicates in all

[11]http://bioportal.bioontology.org
[12]http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library
[13]http://www.ebi.ac.uk/rdf/
[14]http://lpis.csd.auth.gr/ontologies/

	Axioms	**Rules**		**Predicates**			**Facts**
		all	disj	all	horn	disj	
ChEMBL	336	313	12	253	46	14	2.9×10^8
UOBM(n)	261	273	1	186	70	21	$2.6n \times 10^5$

Table 2: Test ontologies and datasets for query answering

the tested Disjunctive Datalog programs were Horn, and so could be queried using a Datalog engine (even if the program itself could not be rewritten).

10.2. Query Answering Experiments

We now consider the task of answering (atomic) queries w.r.t. Disjunctive Datalog programs obtained from OWL ontologies and RDF datasets. The main goal of our experiments is to confirm that rewriting a markable Disjunctive Datalog program into Datalog can lead to improved performance and more robust scalability of query answering, even if the reasoner of choice is highly-optimised for disjunctive reasoning.

It is well-known that finding interesting test ontologies that come with a substantial dataset is a major challenge for the evaluation of query answering. In our case, we had the additional constraints that test ontologies are non-Horn and correspond to a markable program. We could find two interesting test cases satisfying these requirements.

- ChEMBL is a real-world ontology publicly available through the European Bioinformatics Institute (EBI) linked data platform.[15] This ontology is $\mathrm{RL}^{\sqcup}$ and it comes with a large-scale dataset, which is also publicly available. To evaluate scalability, we have used a sampling algorithm based on random walks to extract subsets of the data of increasing size.

- UOBM is a widely-used reasoning benchmark that comes with a manually created ontology and a data generator which produces synthetic datasets according to a parameter n that determines data size [60]. KAON2 does not succeed in rewriting the UOBM ontology into Disjunctive Datalog; hence, we considered its $\mathrm{RL}^{\sqcup}$ fragment in our experiments.

Table 2 summarises our test ontologies and datasets. The table indicates *(i)* the number of ontology axioms; *(ii)* the number of rules in the corresponding Disjunctive Datalog program as well as the number of non-Horn rules; *(iii)* the number of predicates together with the number of Horn and disjunctive IDB predicates; and *(iv)* the number of facts in the accompanying datasets, where in the case of UOBM this number is parametrised by n. We restricted our evaluation to queries over disjunctive predicates; the remaining atomic queries (i.e.,

[15]http://www.ebi.ac.uk/rdf/platform

	Data	DLV				Clingo				RDFox	
		ontology		rewriting		ontology		rewriting		rewriting	
		time	error	time	error	time	error	time	error	time	error
ChEMBL	0.1%	940	8	5	0	29	0	6	0	1	0
	0.4%	—	all	21	0	59	3	23	0	3	0
	0.7%	—	all	36	0	121	4	41	0	6	0
	1%	—	all	53	0	131	5	60	0	9	0
UOBM(n)	$n = 10$	37	2	43	0	327	0	42	0	7	0
	$n = 40$	201	2	221	0	—	all	180	0	32	0
	$n = 70$	429	2	401	0	—	all	335	0	56	0
	$n = 100$	711	2	528	0	—	all	493	0	84	0

Table 3: Impact of rewriting on performance. All times are average times per query in seconds.

those over EDB and Horn predicates) are uninteresting as their computation is not affected by our rewriting techniques.

In our experiments, we considered the well-known Disjunctive Datalog reasoners DLV [5] and Clingo [6] and assessed their scalability on our test ontologies and their corresponding Datalog rewritings. Additionally, we assessed the scalability of the dedicated Datalog reasoner RDFox [10] on the rewritten ontologies.

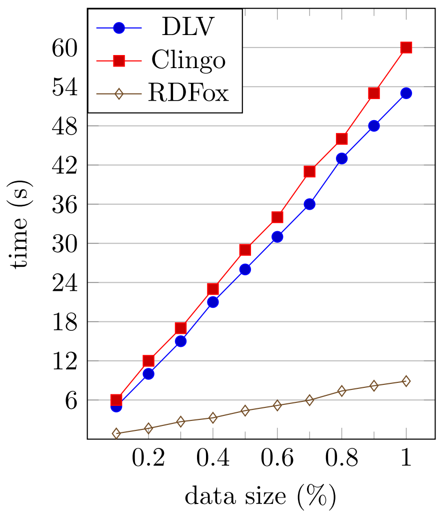
We proceeded as follows for each test ontology $\mathcal{O}$ and reasoning system:

- We normalised the ontology as a Disjunctive Datalog program $\mathcal{P}_{\mathcal{O}}$ and computed its Datalog rewriting $\mathcal{P}'_{\mathcal{O}}$ via transposition.
- For each test dataset $\mathcal{D}$ we used the reasoner to compute the answers to all atomic queries over disjunctive predicates w.r.t. $\mathcal{P}_{\mathcal{O}} \cup \mathcal{D}$ and $\mathcal{P}'_{\mathcal{O}} \cup \mathcal{D}$, respectively. In the case of RDFox, we only considered the latter.

Table 3 summarises our results. The first column indicates the tested ontology and data. In the case of ChEMBL, percentages indicate the size of the data sample w.r.t. the total size of the dataset as in Table 2; in the case of UOBM, the size of the data is determined by the value of the parameter n as in Table 2. For each reasoner, results are split between those obtained for the input ontology ("ontology") and its Datalog rewriting ("rewriting"). Times indicate the average time in seconds needed to answer an atomic query, whereas errors refer to the number of queries for which the 20 minute time-out was exceeded. Additionally, average query answering times for the rewritten ontologies are visualised in Figure 3.

We can observe that the performance of Disjunctive Datalog reasoners consistently improves after rewriting. This is so despite the fact that both DLV and Clingo are highly optimised in their treatment of disjunctions.

As we can see, performance improvements can be very significant. For instance, DLV could not answer any query on ChEMBL for most of our data samples; after rewriting, however, it succeeded in answering all queries in under a minute on average. Similarly, Clingo failed for all queries on UOBM for



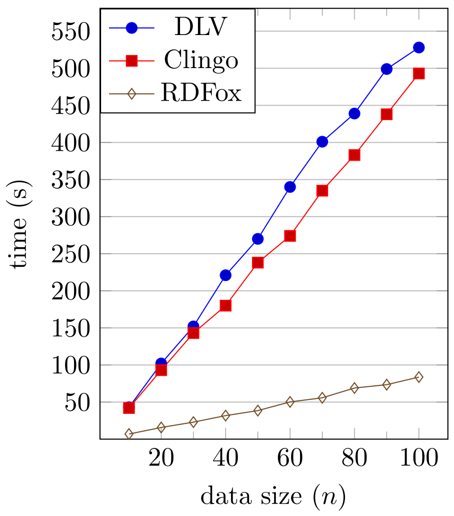


Figure 3: Scalability of query answering on ChEMBL (left) and UOBM (right)

$n \geq 40$, but succeeded on all queries for all datasets after rewriting. Furthermore, we can observe in Figure 3 that query answering times after rewriting increased linearly in data size for all cases.

Finally, our results also suggest that a dedicated Datalog reasoner such as RDFox can offer very favourable scalability when answering queries over large datasets and Datalog programs obtained via transposition. In particular, RDFox was capable of answering all queries for ChEMBL in under 10 seconds on average, even for the largest dataset we considered. Similarly, RDFox could answer all queries in just over a minute, even for UOBM(100).

To sum up, we can draw two main conclusions from our experiments. On the one hand, there is a fair number of ontologies used in practice that correspond (either directly or by means of KAON2-style rewritings) to markable Disjunctive Datalog programs, to which our rewriting techniques are applicable. On the other hand, when the program is markable, rewriting it first into Datalog via transposition can significantly improve reasoning performance and robustness.

11. Conclusion and Future Work

In this paper, we have studied the problem of rewriting Disjunctive Datalog programs and Description Logics ontologies into Datalog.

From a theoretical perspective, we have established a novel characterisation of Datalog rewritability in terms of linearisability: a Disjunctive Datalog program is Datalog rewritable if and only if it can be rewritten into a linear program. Our characterisation relies on the correctness of program transposition—a novel polynomial transformation where IDB atoms are moved from head and body and vice versa while their predicates are replaced with fresh predicates of higher arity. Motivated by our characterisation and the properties of transposition, we have then proposed the classes of weakly linear and markable Disjunctive Datalog programs and showed that they admit polynomial Datalog rewritings. For programs that do not satisfy the WL or markability conditions, we have proposed a sound but incomplete procedure based on unfolding transformations and proved its correctness. We have shown that our results for Disjunctive Datalog can be naturally applied to DL ontologies as well, and we have proposed classes of ontologies that admit polynomial and exponential rewritings, respectively. Finally, we have extended all our results to conjunctive query answering and identified classes of programs and queries that admit a Datalog rewriting.

From a practical perspective, our techniques enable the use of scalable Datalog engines for data reasoning. This is especially relevant for DL ontologies, where reasoning with large-scale datasets is a major challenge in practice. Our experiments have confirmed the potential applicability of our approach.

We see many interesting directions for future work. In particular, we are currently working on lifting transposition to programs with function symbols; preliminary results are reported in [61]. This would allow us to identify classes of first-order programs (and thus also expressive ontologies) that admit polynomial Datalog rewritings via transposition. Additionally, it would be interesting to devise rewriting techniques that combine the benefits of resolution [2, 28] and transposition. Finally, we believe that our techniques could be exploited to enhance "pay-as-you-go" query answering systems such as PAGOdA [62], where the idea is to delegate the bulk of the computational workload to a Datalog reasoner.

Acknowledgments

This work was supported by the Royal Society, the EPSRC projects Score!, ExODA, MaSI^3 and DBOnto, and the FP7 project Optique. We would like to thank the anonymous referees for their helpful suggestions, which have significantly improved the paper.

References

[1] M. Kaminski, Y. Nenov, B. Cuenca Grau, Datalog rewritability of disjunctive Datalog programs and its applications to ontology reasoning, in: C. E. Brodley, P. Stone (Eds.), Proc. 28th AAAI Conference on Artificial Intelligence, AAAI, 1077–1083, 2014.

[2] M. Kaminski, Y. Nenov, B. Cuenca Grau, Computing Datalog rewritings for disjunctive Datalog programs and description logic ontologies, in: R. Kontchakov, M. Mugnier (Eds.), Web Reasoning and Rule Systems – 8th International Conference, RR 2014, vol. 8741 of *LNCS*, Springer, 76–91, 2014.

[3] T. Eiter, G. Gottlob, H. Mannila, Disjunctive Datalog, ACM Trans. Database Syst. 22 (3) (1997) 364–418.

[4] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.

[5] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (3) (2006) 499–562.

[6] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, S. Thiele, Engineering an incremental ASP solver, in: M. García de la Banda, E. Pontelli (Eds.), Proc. 24th International Conference on Logic Programming, ICLP 2008, vol. 5366 of *LNCS*, Springer, 190–205, 2008.

[7] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL 2 Web Ontology Language Profiles, W3C Recommendation, W3C, 2009.

[8] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, R. Velkov, OWLIM: A family of scalable semantic repositories, Semantic Web 2 (1) (2011) 33–42.

[9] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, J. Srinivasan, Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle, in: G. Alonso, J. A. Blakeley, A. L. P. Chen (Eds.), Proc. 24th International Conference on Data Engineering, ICDE 2008, IEEE, 1239–1248, 2008.

[10] B. Motik, Y. Nenov, R. Piro, I. Horrocks, D. Olteanu, Parallel materialisation of Datalog programs in centralised, main-memory RDF systems, in: C. E. Brodley, P. Stone (Eds.), Proc. 28th AAAI Conference on Artificial Intelligence, AAAI, 129–137, 2014.

[11] G. Gottlob, C. H. Papadimitriou, On the complexity of single-rule Datalog queries, Inf. Comput. 183 (1) (2003) 104–122.

[12] H. Tamaki, T. Sato, Unfold/fold transformation of logic programs, in: S. Tärnlund (Ed.), Proc. 2nd International Logic Programming Conference, Uppsala University, Sweden, 127–138, 1984.

[13] M. Gergatsoulis, Unfold/fold transformations for disjunctive logic programs, Inf. Process. Lett. 62 (1) (1997) 23–29.

[14] C. Lutz, F. Wolter, Non-uniform data complexity of query answering in description logics, in: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), Principles of Knowledge Representation and Reasoning: Proc. 13th International Conference, KR 2012, AAAI, 297–307, 2012.

[15] L. Bachmair, H. Ganzinger, Resolution theorem proving, in: J. A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Elsevier and MIT Press, 19–99, 2001.

[16] F. Bry, N. Eisinger, T. Eiter, T. Furche, G. Gottlob, C. Ley, B. Linse, R. Pichler, F. Wei, Foundations of rule-based query answering, in: G. Antoniou, U. Aßmann, C. Baroglio, S. Decker, N. Henze, P. Patranjan, R. Tolksdorf (Eds.), Reasoning Web 2007, vol. 4636 of *LNCS*, Springer, 1–153, 2007.

[17] J. A. Robinson, Automatic deduction with hyper-resolution, Int. J. Comput. Math. 1 (3) (1965) 227–234.

[18] T. Feder, M. Y. Vardi, The computational structure of monotone monadic SNP and constraint satisfiaction: A study through Datalog and group theory, SIAM J. Comput. 28 (1) (1998) 57–104.

[19] S. A. Cook, An observation on time-storage trade off, J. Comput. Syst. Sci. 9 (3) (1974) 308–316.

[20] K. R. Apt, H. A. Blair, A. Walker, Towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 89–148, 1988.

[21] J. D. Ullman, Principles of Database and Knowledge-Base Systems, Volume I, Computer Science Press, 1988.

[22] M. Bienvenu, B. ten Cate, C. Lutz, F. Wolter, Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP, ACM Tans. Database Syst. 39 (4) (2014) 33.

[23] U. Hustadt, B. Motik, U. Sattler, Reasoning in description logics by a reduction to disjunctive Datalog, J. Autom. Reason. 39 (3) (2007) 351–384.

[24] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[25] B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. F. Patel-Schneider, U. Sattler, OWL 2: The next step for OWL, J. Web Semant. 6 (4) (2008) 309–322.

[26] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for very expressive description logics, Log. J. IGPL 8 (3) (2000) 239–263.

[27] M. Cadoli, L. Palopoli, M. Lenzerini, Datalog and description logics: Expressive power, in: S. Cluet, R. Hull (Eds.), Proc. 6th International Workshop on Database Programming Languages, DBPL-6, vol. 1369 of *LNCS*, Springer, 281–298, 1997.

[28] B. Cuenca Grau, B. Motik, G. Stoilos, I. Horrocks, Computing Datalog rewritings beyond Horn ontologies, in: F. Rossi (Ed.), IJCAI 2013, Proc. 23rd International Joint Conference on Artificial Intelligence, AAAI, 832–838, 2013.

[29] S. Rudolph, M. Krötzsch, P. Hitzler, Type-elimination-based reasoning for the description logic $\mathcal{SHIQ}_{b_s}$ using decision diagrams and disjunctive Datalog, Log. Methods Comput. Sci. 8 (1:12) (2012) 1–37.

[30] M. Krötzsch, S. Rudolph, P. Hitzler, Complexities of Horn description logics, ACM Trans. Comput. Log. 14 (1:2) (2013) 1–36.

[31] R. Ben-Eliyahu-Zohary, L. Palopoli, Reasoning with minimal models: Efficient algorithms and applications, Artif. Intell. 96 (2) (1997) 421–449.

[32] R. Ben-Eliyahu-Zohary, L. Palopoli, V. Zemlyanker, More on tractable disjunctive Datalog, J. Log. Program. 46 (1-2) (2000) 61–101.

[33] M. Gelfond, H. Przymusinska, V. Lifschitz, M. Truszczynski, Disjunctive defaults, in: J. F. Allen, R. Fikes, E. Sandewall (Eds.), Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann, 230–237, 1991.

[34] R. Ben-Eliyahu-Zohary, R. Dechter, Propositional semantics for disjunctive logic programs, Ann. Math. Artif. Intell. 12 (1–2) (1994) 53–87.

[35] J. Dix, G. Gottlob, V. W. Marek, Reducing disjunctive to non-disjunctive semantics by shift-operations, Fundam. Inform. 28 (1–2) (1996) 87–100.

[36] H. Zhang, Y. Zhang, First-order expressibility and boundedness of disjunctive logic programs, in: F. Rossi (Ed.), IJCAI 2013, Proc. 23rd International Joint Conference on Artificial Intelligence, AAAI, 1198–1204, 2013.

[37] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The DL-Lite family, J. Autom. Reason. 39 (3) (2007) 385–429.

[38] S. Heymans, T. Eiter, G. Xiao, Tractable reasoning with DL-programs over Datalog-rewritable description logics, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), ECAI 2010, Proc. 19th European Conference on Artificial Intelligence, vol. 215 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 35–40, 2010.

[39] H. Pérez-Urbina, B. Motik, I. Horrocks, Tractable query answering and rewriting under description logic constraints, J. Appl. Log. 8 (2) (2010) 186–209.

[40] R. Rosati, A. Almatelli, Improving query answering over DL-Lite ontologies, in: F. Lin, U. Sattler, M. Truszczynski (Eds.), Principles of Knowledge Representation and Reasoning: Proc. 12th International Conference, KR 2010, AAAI, 290–300, 2010.

[41] G. Orsi, A. Pieris, Optimizing query answering under ontological constraints, Proc. VLDB Endow. 4 (11) (2011) 1004–1015.

[42] T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, G. Xiao, Query rewriting for Horn-SHIQ plus rules, in: J. Hoffmann, B. Selman (Eds.), Proc. 26th AAAI Conference on Artificial Intelligence, AAAI, 726–733, 2012.

[43] M. Rodríguez-Muro, D. Calvanese, High performance query answering over DL-Lite ontologies, in: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), Principles of Knowledge Representation and Reasoning: Proc. 13th International Conference, KR 2012, AAAI, 308–318, 2012.

[44] T. Venetis, G. Stoilos, G. B. Stamou, Query extensions and incremental query rewriting for OWL 2 QL ontologies, J. Data Semant. 3 (1) (2014) 1–23.

[45] D. Trivela, G. Stoilos, A. Chortaras, G. B. Stamou, Optimising resolution-based rewriting algorithms for OWL ontologies, J. Web Semant. 33 (2015) 30–49.

[46] G. Gottlob, M. Manna, M. Morak, A. Pieris, On the complexity of ontological reasoning under disjunctive existential rules, in: B. Rovan, V. Sassone, P. Widmayer (Eds.), Mathematical Foundations of Computer Science 2012 – 37th International Symposium, MFCS 2012, vol. 7464 of *LNCS*, Springer, 1–18, 2012.

[47] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The DL-Lite family and relations, J. Artif. Intell. Res. 36 (2009) 1–69.

[48] P. Bourhis, M. Morak, A. Pieris, The impact of disjunction on query answering under guarded-based existential rules, in: F. Rossi (Ed.), IJCAI 2013, Proc. 23rd International Joint Conference on Artificial Intelligence, AAAI, 796–802, 2013.

[49] M. Kaminski, B. Cuenca Grau, Sufficient conditions for first-order and Datalog rewritability in $\mathcal{ELU}$, in: T. Eiter, B. Glimm, Y. Kazakov, M. Krötzsch (Eds.), Informal Proc. 26th International Workshop on Description Logics, vol. 1014 of *CEUR Workshop Proceedings*, CEUR-WS.org, 271–293, 2013.

[50] F. Bancilhon, D. Maier, Y. Sagiv, J. D. Ullman, Magic sets and other strange ways to implement logic programs, in: A. Silberschatz (Ed.), Proc. 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, ACM, 1–15, 1986.

[51] S. Greco, Binding propagation techniques for the optimization of bound disjunctive queries, IEEE Trans. Knowl. Data Eng. 15 (2) (2003) 368–385.

[52] M. Alviano, W. Faber, G. Greco, N. Leone, Magic sets for disjunctive Datalog programs, Artif. Intell. 187–188 (2012) 156–192.

[53] W. Faber, S. Woltran, Manifold answer-set programs and their applications, in: M. Balduccini, T. C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to

Michael Gelfond on the Occasion of His 65th Birthday, vol. 6565 of *LNCS*, Springer, 44–63, 2011.

[54] F. Afrati, M. Gergatsoulis, F. Toni, Linearisability of Datalog programs, Theor. Comput. Sci. 308 (1–3) (2003) 199–226.

[55] M. Proietti, A. Pettorossi, The Loop absorption and the generalization strategies for the development of logic programs and partial deduction, J. Log. Program. 16 (1) (1993) 123–161.

[56] B. Aspvall, M. F. Plass, R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Inf. Process. Lett. 8 (3) (1979) 121–123.

[57] B. Motik, Reasoning in Description Logics using Resolution and Deductive Databases, Ph.D. thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, 2006.

[58] T. Gardiner, D. Tsarkov, I. Horrocks, Framework for an automated comparison of description logic reasoners, in: I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.), Proc. 5th International Semantic Web Conference, ISWC 2006, vol. 4273 of *LNCS*, Springer, 654–667, 2006.

[59] E. Sirin, B. Cuenca Grau, B. Parsia, From Wine to Water: Optimizing Description Logic Reasoning for Nominals, in: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006, 90–99, 2006.

[60] L. Ma, Y. Yang, Z. Qiu, G. T. Xie, Y. Pan, S. Liu, Towards a complete OWL ontology benchmark, in: Y. Sure, J. Domingue (Eds.), Proc. 3rd European Semantic Web Conference, ESWC 2006, vol. 4011 of *LNCS*, Springer, 125–139, 2006.

[61] M. Kaminski, B. Cuenca Grau, Computing Horn rewritings of description logics ontologies, in: Q. Yang, M. Wooldridge (Eds.), IJCAI 2015, Proc. 24th International Joint Conference on Artificial Intelligence, AAAI, 3091–3097, 2015.

[62] Y. Zhou, Y. Nenov, B. Cuenca Grau, I. Horrocks, Pay-as-you-go OWL query answering using a triple store, in: C. E. Brodley, P. Stone (Eds.), Proc. 28th AAAI Conference on Artificial Intelligence, AAAI, 1142–1148, 2014.