Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Datalog Rewritability of Disjunctive Datalog Programs and Its Applications to Ontology Reasoning

Mark Kaminski and **Yavor Nenov** and **Bernardo Cuenca Grau**
Department of Computer Science, University of Oxford, UK

Abstract

We study the problem of rewriting a disjunctive datalog program into plain datalog. We show that a disjunctive program is rewritable if and only if it is equivalent to a linear disjunctive program, thus providing a novel characterisation of datalog rewritability. Motivated by this result, we propose *weakly linear disjunctive datalog*—a novel rule-based KR language that extends both datalog and linear disjunctive datalog and for which reasoning is tractable in data complexity. We then explore applications of weakly linear programs to ontology reasoning and propose a tractable extension of OWL 2 RL with disjunctive axioms. Our empirical results suggest that many non-Horn ontologies can be reduced to weakly linear programs and that query answering over such ontologies using a datalog engine is feasible in practice.

1 Introduction

Disjunctive datalog, which extends plain datalog by allowing disjunction in the head of rules, is a prominent KR formalism that has found many applications in the areas of deductive databases, information integration and ontological reasoning (Eiter, Gottlob, and Mannila 1997; Dantsin et al. 2001).[1] Disjunctive datalog is a powerful language, which can model incomplete information. Expressiveness comes, however, at the expense of computational cost: fact entailment is co-NEXPTIME-complete in combined complexity and co-NP-complete w.r.t. data (Eiter, Gottlob, and Mannila 1997). Thus, even with the development of optimised implementations (Leone et al. 2006), robust behaviour of reasoners in data-intensive applications cannot be guaranteed.

Plain datalog offers more favourable computational properties (EXPTIME-completeness in combined complexity and PTIME-completeness w.r.t. data) at the expense of a loss in expressive power (Dantsin et al. 2001). Tractability in data complexity is an appealing property for data-intensive KR; in particular, the RL profile of the ontology language OWL 2 was designed such that each ontology corresponds to a datalog program (Motik et al. 2009). Furthermore, datalog programs obtained from RL ontologies contain rules of a restricted shape, and they 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 OWLim (Bishop et al. 2011), Oracle's Semantic Data Store (Wu et al. 2008), and RDFox (Motik et al. 2014).

We study the problem of rewriting a disjunctive datalog program into an equivalent datalog program (i.e., one that entails the same facts for every dataset). By computing such rewritings, we can ensure tractability w.r.t. data and exploit reasoning infrastructure available for datalog. Not every disjunctive datalog program is, however, datalog rewritable (Afrati, Cosmadakis, and Yannakakis 1995).

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 (Dantsin et al. 2001). For disjunctive programs the effects of the linearity restriction are, to the best of our knowledge, unknown. In Section 3, we show that every linear disjunctive program can be polynomially transformed into an equivalent datalog program; conversely, we also provide a polynomial transformation from datalog into linear disjunctive datalog. Thus, linear disjunctive datalog and datalog have the same computational properties, and linearisability of disjunctive programs is equivalent to rewritability into datalog.

Motivated by our characterisation, in Section 4 we propose *weakly linear disjunctive datalog*: a rule language that extends both datalog and linear disjunctive datalog. In a weakly linear (WL for short) program, the linearity requirement is relaxed: instead of applying to all IDB predicates, it applies only to those that "depend" on a disjunctive rule. Analogously to linear disjunctive programs, WL programs can be polynomially rewritten into datalog. Thus, our language captures disjunctive information while leaving the favourable computational properties of datalog intact.

In Section 5, we propose a linearisation procedure based on unfolding transformations. Our procedure picks a non-WL rule and a "culprit" body atom and replaces it with WL rules by "unfolding" the selected atom. Our procedure is incomplete: if it succeeds, it outputs a WL program, which is rewritten into datalog; if it fails, no conclusion can be drawn.

[1] Disjunctive datalog typically allows for negation-as-failure, which we don't consider since we focus on monotonic reasoning.

In Section 6, we focus on ontology reasoning. We propose an extension of OWL 2 RL with disjunctive axioms such that each ontology in our extended profile maps to a WL program. We show that the resulting programs can be evaluated in polynomial time in *combined* complexity; thus, fact entailment in our language is no harder than in OWL 2 RL. Finally, we argue that the algorithm in (Hustadt, Motik, and Sattler 2007) can be combined with our techniques to rewrite a $\mathcal{SHIQ}$ ontology into a plain datalog program.

We have evaluated our techniques on a large ontology repository. Our results show that many non-Horn ontologies can be rewritten into WL programs, and thus into datalog. We have tested the scalability of query answering using our approach, with promising results. Proofs are delegated to an extended version (see `arXiv:1404.3141`).

2 Preliminaries

We use standard first-order syntax and semantics and assume all formulae to be function-free. We assume that equality $\approx$ is an ordinary predicate and that every set of set of formulae contains the standard explicit axiomatisation of $\approx$ as a congruence relation for its signature.

A *fact* is a ground atom and a *dataset* is a finite set of facts. A *rule* r is a sentence 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 conjunction of distinct equality-free atoms, and $\psi(\vec{x})$ is a disjunction of distinct atoms. Formula φ is the *body* of r, and ψ is the *head*. Quantifiers in rules are omitted. We assume that rules are *safe*, i.e., all variables in the head occur in the body. A rule is *datalog* if $\psi(\vec{x})$ has at most one atom, and it is *disjunctive* otherwise. A *program* $\mathcal{P}$ is a finite set of rules; it is *datalog* if it consists only of datalog rules, and *disjunctive* otherwise. We assume that rules in $\mathcal{P}$ do not share variables.

For convenience, we treat $\top$ and $\bot$ in a non-standard way as a unary and a nullary predicate, respectively. Given a program $\mathcal{P}$, $\mathcal{P}_\top$ is the program with a rule $Q(x_1,\ldots,x_n) \rightarrow \top(x_i)$ for each predicate Q in $\mathcal{P}$ and each $1 \leq i \leq n$, and a rule $\rightarrow \top(a)$ for each constant a in $\mathcal{P}$. We assume that $\mathcal{P}_\top \subseteq \mathcal{P}$ and $\top$ does not occur in head position in $\mathcal{P} \setminus \mathcal{P}_\top$. We define $\mathcal{P}_\bot$ as consisting of a rule with $\bot$ as body and empty head. We assume $\mathcal{P}_\bot \subseteq \mathcal{P}$ and no rule in $\mathcal{P} \setminus \mathcal{P}_\bot$ has an empty head or $\bot$ in the body. Thus, $\mathcal{P} \cup \mathcal{D} \models \top(a)$ for every a in $\mathcal{P} \cup \mathcal{D}$, and $\mathcal{P} \cup \mathcal{D}$ is unsatisfiable iff $\mathcal{P} \cup \mathcal{D} \models \bot$.

Head predicates in $\mathcal{P} \setminus \mathcal{P}_\top$ are *intensional* (or *IDB*) in $\mathcal{P}$. All other predicates (including $\top$) are *extensional* (*EDB*). An atom is intensional (extensional) if so is its predicate. A rule is *linear* if it has at most one IDB body atom. A program $\mathcal{P}$ is linear if all its rules are. In contrast to KR, in logic programming it is often assumed that IDB predicates do not occur in datasets. This assumption can be lifted (see, e.g., (Bry et al. 2007)): for every $\mathcal{P}$ and IDB predicate Q in $\mathcal{P}$, let Q' be a fresh predicate; the *IDB expansion* $\mathcal{P}^e$ of $\mathcal{P}$ is obtained from $\mathcal{P}$ by renaming each IDB predicate Q in $\mathcal{P}$ with Q' and adding a rule $Q(\vec{x}) \rightarrow Q'(\vec{x})$, with $\vec{x}$ distinct variables. Then, for each $\mathcal{D}$ and each fact α over the signature of $\mathcal{P}$ we have $\mathcal{P} \cup \mathcal{D} \models \alpha$ iff $\mathcal{P}^e \cup \mathcal{D} \models \alpha\theta$, where θ is the predicate substitution mapping each IDB predicate Q to Q'.

The *evaluation* of $\mathcal{P}$ over a dataset $\mathcal{D}$ is the set $\mathsf{Eval}(\mathcal{P},\mathcal{D})$ which comprises $\bot$ if $\mathcal{P} \cup \mathcal{D}$ is unsatisfiable and all facts entailed by $\mathcal{P} \cup \mathcal{D}$ otherwise. For a set of predicates S, $\mathsf{Eval}(\mathcal{P},\mathcal{D})|_S$ consists of those facts in $\mathsf{Eval}(\mathcal{P},\mathcal{D})$ involving predicates in $S \cup \{\bot\}$. Program $\mathcal{P}'$ is a *rewriting* of $\mathcal{P}$ w.r.t. a set of predicates S if there is an injective predicate renaming θ such that $(\mathsf{Eval}(\mathcal{P},\mathcal{D})|_S)\theta = \mathsf{Eval}(\mathcal{P}',\mathcal{D})|_{S\theta}$ for every dataset $\mathcal{D}$ over the signature of $\mathcal{P}$. The program $\mathcal{P}'$ is a rewriting of $\mathcal{P}$ if $\mathcal{P}'$ is a rewriting of $\mathcal{P}$ w.r.t. the set of all predicates in $\mathcal{P}$. Clearly, $\mathcal{P}^e$ is a rewriting of $\mathcal{P}$.

3 Characterisation of Datalog Rewritability

In this section, we establish a strong correspondence between linear disjunctive datalog and plain datalog. We show that every linear disjunctive program can be polynomially rewritten into datalog and, conversely, every datalog program is polynomially rewritable to a linear disjunctive program. Consequently, we not only can conclude that fact entailment over linear programs has exactly the same data and combined complexity as over plain datalog programs, but also that a disjunctive program is datalog rewritable if and only if it is linearisable. Thus, datalog rewritability and linearisability of disjunctive programs are equivalent problems.

From Linear Programs to Datalog We first show that linear disjunctive programs can be polynomially rewritten into datalog. Let us consider the following program $\mathcal{P}_1$, which we want to rewrite into a datalog program $\Xi(\mathcal{P}_1)$:

$$\mathcal{P}_1 = \{\, V(x) \rightarrow B(x) \vee G(x) \quad (1)$$

$$G(y) \wedge E(x,y) \rightarrow B(x) \quad (2)$$

$$B(y) \wedge E(x,y) \rightarrow G(x) \,\} \quad (3)$$

Predicates V and E are EDB, so their extension depends solely on $\mathcal{D}$. To prove facts about IDB predicates G and B we introduce fresh binary predicates B^G, B^B, G^B, and G^G. Intuitively, if a fact $B^G(c,d)$ holds in $\Xi(\mathcal{P}_1) \cup \mathcal{D}$ then proving $B(c)$ suffices for proving $G(d)$ in $\mathcal{P}_1 \cup \mathcal{D}$. To "initialise" the extension of these fresh predicates we need rules $\top(x) \rightarrow X^X(x,x)$ with $X \in \{G,B\}$. The key step is then to "flip" the direction of all rules in $\mathcal{P}_1$ involving G or B 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. Thus, Rule (2) leads to the following rules in $\Xi(\mathcal{P}_1)$ for each IDB predicate X:

$$B^X(x,z) \wedge E(x,y) \rightarrow G^X(y,z)$$

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 $E(x,y)$ holds, 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 "flip" this rule as follows, with X IDB:

$$V(x) \wedge B^X(x,z) \wedge G^X(x,z) \rightarrow X(z)$$

Similarly to the previous case, this rule follows from Rule (1): if $V(x)$ 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, we introduce rules that allow us to derive facts about the IDB predicates G and B from facts derived about the auxiliary predicates. For example, the rule $B(x) \wedge B^X(x,z) \rightarrow X(z)$ states that if $B(x)$ holds and is sufficient to prove $X(z)$, then $X(z)$ must also hold.

Definition 1. Let $\mathcal{P}$ be a linear program and let Σ be the set of IDB predicates in $\mathcal{P} \setminus \mathcal{P}_\top$. For each $(P, Q) \in \Sigma^2$, let P^Q be a fresh predicate unique to (P, Q) where $\mathsf{arity}(P^Q) = \mathsf{arity}(P) + \mathsf{arity}(Q)$. Then $\Xi(\mathcal{P})$ is the datalog program containing the rules given next, where φ is the conjunction of all EDB atoms in a rule, $\varphi_\top$ is the least conjunction of $\top$-atoms needed to make a rule safe, all predicates P_i are in Σ, and $\vec{y}$, $\vec{z}$ are disjoint vectors of distinct fresh variables:

1. a rule $\varphi_\top \to R^R(\vec{y}, \vec{y})$ for every $R \in \Sigma$;
2. a rule $\varphi_\top \wedge \varphi \wedge \bigwedge_{i=1}^n P_i^R(\vec{s}_i, \vec{y}) \to Q^R(\vec{t}, \vec{y})$ for every rule $\varphi \wedge Q(\vec{t}) \to \bigvee_{i=1}^n P_i(\vec{s}_i) \in \mathcal{P} \setminus \mathcal{P}_\top$ and every $R \in \Sigma$;
3. a rule $\varphi \wedge \bigwedge_{i=1}^n P_i^R(\vec{s}_i, \vec{y}) \to R(\vec{y})$ for every rule $\varphi \to \bigvee_{i=1}^n P_i(\vec{s}_i) \in \mathcal{P} \setminus \mathcal{P}_\top$ and every $R \in \Sigma$;
4. a rule $Q(\vec{z}) \wedge Q^R(\vec{z}, \vec{y}) \to R(\vec{y})$ for every $(Q, R) \in \Sigma^2$. $\diamond$

This transformation is quadratic and the arity of predicates is at most doubled. For $\mathcal{P}_1$, we obtain the following datalog program, where each rule mentioning X stands for one rule where $X = B$ and one where $X = G$:

$$\Xi(\mathcal{P}_1) = \{\, V(x) \wedge B^X(x, z) \wedge G^X(x, z) \to X(z) \quad (1')$$
$$B^X(x, z) \wedge E(x, y) \to G^X(y, z) \quad (2')$$
$$G^X(x, z) \wedge E(x, y) \to B^X(y, z) \quad (3')$$
$$\top(x) \to X^X(x, x) \quad (4)$$
$$B(x) \wedge B^X(x, z) \to X(z) \quad (5)$$
$$G(x) \wedge G^X(x, z) \to X(z) \,\} \quad (6)$$

Correctness of Ξ is established by the following theorem.

Theorem 2. *If $\mathcal{P}$ is linear, then $\Xi(\mathcal{P})$ is a polynomial datalog rewriting of $\mathcal{P}$.*

Thus, fact entailment over linear programs is no harder than in datalog: PTIME w.r.t. data and EXPTIME in combined complexity. Formally, Theorem 2 is shown by induction on hyperresolution derivations of facts entailed by the rules in $\mathcal{P}$ from a given dataset $\mathcal{D}$ (see Appendix). We next sketch the intuitions on $\mathcal{P}_1$ and $\mathcal{D}_1 = \{V(a), V(b), V(c), E(a, b), E(b, c), E(a, c)\}$.

Figure 1, Part (a) shows a linear (hyperresolution) derivation ρ_1 of $B(a)$ from $\mathcal{P}_1 \cup \mathcal{D}_1$ while Part (b) shows a derivation ρ_2 of the same fact from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$. We represent derivations as trees whose nodes are labeled with disjunctions of facts and where every inner node is derived from its children using a rule of the program (initialisation rules in ρ_2 are omitted for brevity). We first show that if $B(a)$ is provable in $\mathcal{P}_1 \cup \mathcal{D}_1$, then it is entailed by $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$. From the premise, a linear derivation such as ρ_1 exists. The crux of the proof is to show that each disjunction of facts in ρ_1 corresponds to a set of facts over the auxiliary predicates entailed by $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$. Furthermore, these facts must be of the form $X^B(u, a)$, 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 $B^B(c, a)$ and $G^B(c, a)$, which are provable from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$, as witnessed by ρ_2. Since ρ_1 is linear, it has a unique rule application that has only EDB atoms as premises, i.e., the application of (1), which generates $B(c) \vee G(c)$. Since $B^B(c, a)$ and $G^B(c, a)$ are provable from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$, we can apply (1') to derive $B(a)$.

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

From Datalog to Linear Programs The transformation from datalog to linear disjunctive datalog is based on the same ideas, but it is simpler in that we no longer distinguish between EDB and IDB atoms: a rule in $\mathcal{P}$ is now "flipped" by moving *all* its atoms from the head to the body and vice-versa. Moreover, we make use of the IDB expansion $\mathcal{P}^e$ of $\mathcal{P}$ to ensure linearity of the resulting disjunctive program.

Definition 3. Let $\mathcal{P}$ be a datalog program. For each pair (P, Q) of predicates in $\mathcal{P}$, let P^Q be a fresh predicate unique to (P, Q) where $\mathsf{arity}(P^Q) = \mathsf{arity}(P) + \mathsf{arity}(Q)$. Furthermore, let $\mathcal{P}^e$ be the IDB expansion of $\mathcal{P}$. Then, $\Psi(\mathcal{P})$ is the linear disjunctive program containing, for each IDB predicate R in $\mathcal{P}^e$ the rules given next, where $\varphi_\top$ is the least conjunction of $\top$-atoms making a rule safe and $\vec{y} = y_1 \dots y_{\mathsf{arity}(R)}$ is a vector of distinct fresh variables:

1. a rule $\varphi_\top \wedge Q^R(\vec{t}, \vec{y}) \to \bigvee_{i=1}^n P_i^R(\vec{s}_i, \vec{y})$ for every rule $\bigwedge_{i=1}^n P_i(\vec{s}_i) \to Q(\vec{t}) \in \mathcal{P}^e \setminus \mathcal{P}^e_\top$, where $Q(\vec{t}) \neq \bot$ and $\bigvee_{i=1}^n P_i^R(\vec{s}_i, \vec{y})$ is interpreted as $\bot$ if $n = 0$;
2. a rule $\varphi_\top \to \bigvee_{i=1}^n P_i^R(\vec{s}_i, \vec{y})$ for every $\bigwedge_{i=1}^n P_i(\vec{s}_i) \to \bot \in \mathcal{P}^e \setminus \mathcal{P}^e_\top$;
3. a rule $\varphi_\top \to R^R(\vec{y}, \vec{y})$;
4. a rule $Q(\vec{z}) \wedge Q^R(\vec{z}, \vec{y}) \to R(\vec{y})$ for every EDB predicate Q in $\mathcal{P}^e$, where $\vec{z}$ is a vector of distinct fresh variables. $\diamond$

Again, the transformation is quadratic and the arity of predicates is at most doubled.

Example 4. Consider $\mathcal{P}_2$, which encodes path system accessibility (a canonical PTIME-complete problem):

$$\mathcal{P}_2 = \{\, R(x, y, z) \wedge A(y) \wedge A(z) \to A(x) \,\} \quad (7)$$

Linear datalog is NLOGSPACE, and cannot capture $\mathcal{P}_2$. However, we can rewrite $\mathcal{P}_2$ into linear disjunctive datalog:

$$\Psi(\mathcal{P}_2) = \{\, \top(y) \wedge \top(z) \wedge A'^{A'}(x, u) \quad (7')$$
$$\to R^{A'}(x, y, z, u) \vee A'^{A'}(y, u) \vee A'^{A'}(z, u)$$
$$A'^{A'}(x, y) \to A^{A'}(x, y) \quad (8)$$
$$\top(x) \to A'^{A'}(x, x) \quad (9)$$
$$A(x) \wedge A^{A'}(x, y) \to A'(y) \quad (10)$$
$$R(x, y, z) \wedge R^{A'}(x, y, z, u) \to A'(u) \,\} \quad (11)$$

Rule (7) yields Rule (7') in $\Psi(\mathcal{P}_2)$. Rule (8) is obtained from $A(x) \to A'(x) \in \mathcal{P}^e$. To see why we need the IDB expansion $\mathcal{P}^e$, suppose we replaced $\mathcal{P}^e$ by $\mathcal{P}$ in Definition 3. Rule (8) would not be produced and A' would be replaced by A elsewhere. Then the rule $A(x) \wedge A^A(x, y) \to A(y)$ would not be linear since both A and A^A would be IDB. $\diamond$

Correctness of Ψ is established by the following theorem.

Theorem 5. *If $\mathcal{P}$ is datalog, then $\Psi(\mathcal{P})$ is a polynomial rewriting of $\mathcal{P}$ into a linear disjunctive program.*

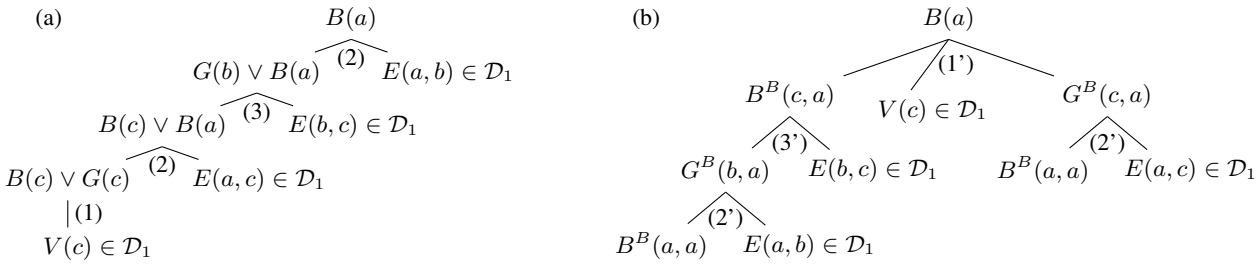


Figure 1: (a) derivation of $B(a)$ from $\mathcal{P}_1 \cup \mathcal{D}_1$; (b) derivation of $B(a)$ from $\Xi(\mathcal{P}_1) \cup \mathcal{D}_1$

From Theorems 2 and 5 we obtain the following results.

Corollary 6. *A disjunctive program $\mathcal{P}$ is datalog rewritable iff it is rewritable into a linear disjunctive program.*

Corollary 7. *Checking $\mathcal{P} \cup \mathcal{D} \models \alpha$ for $\mathcal{P}$ a linear program, $\mathcal{D}$ a dataset and α a fact is* PTIME*-complete w.r.t. data complexity and* EXPTIME*-complete w.r.t. combined complexity.*

4 Weakly Linear Disjunctive Datalog

In this section, we introduce weakly linear programs: a new class of disjunctive datalog programs that extends both datalog and linear disjunctive datalog. The main idea is simple: instead of requiring the body of each rule 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. This intuition is formalised as given next.

Definition 8. The *dependency graph* $G_{\mathcal{P}} = (V, E, \mu)$ of a program $\mathcal{P}$ is the smallest edge-labeled digraph such that:

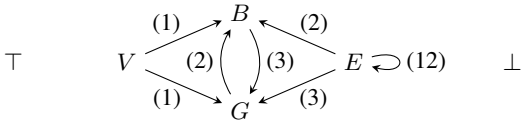
1. V contains every predicate occurring in $\mathcal{P}$;
2. $r \in \mu(P, Q)$ whenever $P, Q \in V$, $r \in \mathcal{P} \setminus \mathcal{P}_\top$, P occurs in the body of r, and Q occurs in the head of r; and
3. $(P, Q) \in E$ whenever $\mu(P, Q)$ is nonempty.

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 *datalog* if it only depends on datalog rules; otherwise, Q is *disjunctive*. Program $\mathcal{P}$ is *weakly linear* (WL for short) if every rule in $\mathcal{P}$ has at most one occurrence of a disjunctive predicate in the body. $\diamond$

Checking whether $\mathcal{P}$ is WL is clearly feasible in polynomial time. If $\mathcal{P}$ is datalog, then all its predicates are datalog and $\mathcal{P}$ is WL. Furthermore, every disjunctive predicate is IDB and hence every linear program is also WL. There are, however, WL programs that are neither datalog nor linear. Consider $\mathcal{P}_3$, which extends $\mathcal{P}_1$ with the following rule:

$$E(y, x) \rightarrow E(x, y) \tag{12}$$

Since E is IDB in $\mathcal{P}_3$, Rules (2) and (3) have two IDB atoms. Thus, $\mathcal{P}_3$ is not linear. The graph $G_{\mathcal{P}_3}$ looks as follows.



Predicate V is EDB and hence does not depend on any rule. Predicates B and G depend on Rule (1) and hence are disjunctive. Finally, predicate E depends only on Rule (12) and hence it is a datalog predicate. Thus, $\mathcal{P}_3$ is WL.

Definition 9. For $\mathcal{P}$ WL, let $\Xi'(\mathcal{P})$ be defined as $\Xi(\mathcal{P})$ in Definition 1 but where: *(i)* Σ is the set of all disjunctive predicates in $\mathcal{P} \setminus \mathcal{P}_\top$; *(ii)* φ denotes the conjunction of all datalog atoms in a rule; and *(iii)* in addition to rules (1)–(4), $\Xi'(\mathcal{P})$ contains every rule in $\mathcal{P}$ with no disjunctive predicates. $\diamond$

By adapting the proof of Theorem 2 we obtain:

Theorem 10. *If $\mathcal{P}$ is WL, then $\Xi'(\mathcal{P})$ is a polynomial datalog rewriting of $\mathcal{P}$.*

Thus, fact entailment over WL programs has the same data and combined complexity as for datalog. Furthermore, $\Xi'(\mathcal{P})$ is a rewriting of $\mathcal{P}$ and hence it preserves the extension of all predicates. If, however, we want to query a specific predicate Q, we can compute a smaller program, which is linear in the size of $\mathcal{P}$ and preserves the extension of Q. Indeed, if Q is datalog, each proof in $\mathcal{P}$ of a fact about Q involves only datalog rules, and if Q is disjunctive, each such proof involves only auxiliary predicates X^Q. Thus, in Ξ' we can dispense with all rules involving auxiliary predicates X^R for $R \neq Q$. In particular, if Q is datalog, the rewriting contains no auxiliary predicates.

Theorem 11. *Let $\mathcal{P}$ be WL, S a set of predicates in $\mathcal{P}$, and $\mathcal{P}'$ obtained from $\Xi'(\mathcal{P})$ by removing all rules with a predicate X^R for $R \notin S$. Then $\mathcal{P}'$ is a rewriting of $\mathcal{P}$ w.r.t. S.*

5 Rewriting Programs via Unfolding

Although WL programs can be rewritten into datalog, not all datalog rewritable programs are WL. Let $\mathcal{P}_4$ be as follows:

$$\mathcal{P}_4 = \{\, A(x) \wedge B(x) \rightarrow C(x) \vee D(x) \tag{13}$$

$$E(x) \rightarrow A(x) \vee F(x) \tag{14}$$

$$C(x) \wedge R(x, y) \rightarrow B(y) \,\} \tag{15}$$

Program $\mathcal{P}_4$ is not WL since both body atoms in (13) are disjunctive. However, $\mathcal{P}_4$ is datalog rewritable.

We now present a rewriting procedure that combines our results in Section 4 with the work of Gergatsoulis (1997) on program transformation for disjunctive logic programs. Our procedure iteratively eliminates non-WL rules by "unfolding" the culprit atoms w.r.t. the other rules in the program. It stops when the program becomes WL, and outputs a datalog program as in Section 4. The procedure is sound: if it

Procedure 1 Rewrite

Input: $\mathcal{P}$: a disjunctive program
Output: a datalog rewriting of $\mathcal{P}$

1: $\mathcal{P}' := \mathcal{P}^e$
2: **while** $\mathcal{P}'$ not WL **do**
3: select $r \in \mathcal{P}'$ with more than one disjunctive body atom
4: select a disjunctive body atom $\alpha \in r$
5: $\mathcal{P}' := \mathsf{Unfold}(\mathcal{P}', r, \alpha)$
6: **return** $\Xi'(\mathcal{P}')$

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 some programs obtained from realistic ontologies can be rewritten into datalog after a few unfolding steps.

Unfolding We start by recapitulating (Gergatsoulis 1997). Given a disjunctive 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 of $\mathcal{P}$ that replaces r with a set of resolvents of r with rules in $\mathcal{P}$ at α (see Appendix). We denote the resulting program by $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$. Unfolding preserves all entailed disjunctions φ of facts: $\mathcal{P} \models \varphi$ iff $\mathsf{Unfold}(\mathcal{P}, r, \alpha) \models \varphi$ for all $\mathcal{P}$, r, α, and φ. However, to ensure that unfolding produces a rewriting we need a stronger correctness result that is dataset independent.

Theorem 12. *Let $\mathcal{P}_0$ be a disjunctive program and let $\mathcal{P}$ be a rewriting of $\mathcal{P}_0$ such that no IDB predicate in $\mathcal{P}$ occurs in $\mathcal{P}_0$. Let r be a rule in $\mathcal{P}$ and α be an IDB body atom of r. Then $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$ is a rewriting of $\mathcal{P}_0$. Moreover, no IDB predicate in $\mathsf{Unfold}(\mathcal{P}, r, \alpha)$ occurs in $\mathcal{P}_0$.*

The Rewriting Procedure Procedure 1 attempts to eliminate rules with several disjunctive body atoms by unfolding one such atom. Note that to satisfy the premise of Theorem 12, unfolding is applied to $\mathcal{P}^e$ rather than $\mathcal{P}$. Correctness of Procedure 1 is established by the following theorem.

Theorem 13. *Let $\mathcal{P}$ be a disjunctive program. If* Rewrite *terminates on $\mathcal{P}$ with output $\mathcal{P}'$, then $\mathcal{P}'$ is a rewriting of $\mathcal{P}$.*

Rewrite first transforms our example program $\mathcal{P}_4$ to

$$\mathcal{P}'_4 = \{\, A'(x) \wedge B'(x) \to C'(x) \vee D'(x) \tag{16}$$
$$E(x) \to A'(x) \vee F'(x) \tag{17}$$
$$C'(x) \wedge R(x,y) \to B'(y) \,\} \cup \mathcal{P}_{\text{aux}} \tag{18}$$

where $\mathcal{P}_{\text{aux}} = \{\, P(x) \to P'(x) \mid P \in \{A, B, C, D, F\} \,\}$ and A', B', C', D', F' are fresh. Rule (16) is not WL in $\mathcal{P}'_4$, and needs to be unfolded. We choose to unfold (16) on $A'(x)$. Thus, in Step 5, Rule (16) is replaced by the rules

$$A(x) \wedge B'(x) \to C'(x) \vee D'(x) \tag{19}$$
$$E(x) \wedge B'(x) \to C'(x) \vee D'(x) \vee F'(x) \tag{20}$$

The resulting $\mathcal{P}''_4$ is WL, and Rewrite returns $\Xi'(\mathcal{P}''_4)$.

6 Application to OWL Ontologies

The RL profile is a fragment of OWL 2 for which reasoning is tractable and practically realisable by means of rule-based technologies. RL is also a fragment of datalog: each RL ontology can be normalised to a datalog program.

1.	$A \sqsubseteq \leq 1\, R.B$	$A(z) \wedge R(z,x_1) \wedge B(x_1) \wedge R(z,x_2) \wedge B(x_2) \to x_1 \approx x_2$
2.	$A \sqcap B \sqsubseteq C$	$A(x) \wedge B(x) \to C(x)$
3.	$\exists R.A \sqsubseteq B$	$R(x,y) \wedge A(y) \to B(x)$
4.	$R \sqsubseteq S$	$R(x_1,x_2) \to S(x_1,x_2)$
5.	$R \circ S \sqsubseteq T$	$R(x_1,z) \wedge S(z,x_2) \to T(x_1,x_2)$
6.	$A \sqsubseteq \mathsf{Self}(R)$	$A(x) \to R(x,x)$
7.	$\mathsf{Self}(R) \sqsubseteq A$	$R(x,x) \to A(x)$
8.	$R \sqsubseteq S^-$	$R(x,y) \to S(y,x)$
9.	$A \sqsubseteq \{a\}$	$A(x) \to x \approx a$
10.	$\{a\} \sqsubseteq A$	$A(a)$
11.	$A \sqsubseteq B \sqcup C$	$A(x) \to B(x) \vee C(x)$

Table 1: Normalised $\text{RL}^{(\sqcup)}$ axioms, with A, B atomic or $\top$, C atomic or $\bot$, R, S, T atomic roles, a an individual.

We next show how to extend RL with disjunctions while retaining tractability of consistency checking and fact entailment in *combined complexity*. We first recapitulate the kinds of normalised axioms that can occur in an RL ontology. We assume familiarity with Description Logic (DL) notation.

A (normalised) *RL ontology* is a finite set of DL axioms of the form 1-10 in Table 1. The table also provides the translation of DL axioms into rules. We define $\text{RL}^{\sqcup}$ as the extension of RL with axioms capturing disjunctive knowledge.

Definition 14. An *$\text{RL}^{\sqcup}$ ontology* is a finite set of DL axioms of the form 1-11 in Table 1. ◇

Fact entailment in $\text{RL}^{\sqcup}$ is co-NP-hard since $\text{RL}^{\sqcup}$ can encode non-3-colourability. Membership in co-NP holds since rules have bounded number of variables, and hence programs can be grounded in polynomial time (see Appendix). Tractability can be regained if we restrict ourselves to $\text{RL}^{\sqcup}$ ontologies corresponding to WL programs. WL programs $\mathcal{P}$ obtained from $\text{RL}^{\sqcup}$ ontologies have bounded number of variables, and thus variables in $\Xi'(\mathcal{P})$ are also bounded.

Theorem 15. *Checking $\mathcal{O} \cup \mathcal{D} \models \alpha$, for $\mathcal{O}$ an $\text{RL}^{\sqcup}$ ontology that corresponds to a WL program, is* PTIME*-complete.*

Thus, fact entailment in $\text{RL}^{\sqcup}$ is no harder than in RL, and one can use scalable engines such as RDFox. Our experiments indicate that many ontologies captured by $\text{RL}^{\sqcup}$ are either WL or can be made WL via unfolding, which makes data reasoning over such ontologies feasible.[2]

Dealing with Expressive Ontology Languages Hustadt, Motik, and Sattler (2007) developed an algorithm for transforming $\mathcal{SHIQ}$ ontologies into an equivalent disjunctive datalog program. Cuenca Grau et al. (2013) combined this algorithm with a knowledge compilation procedure (called Compile-Horn) obtaining a sound but incomplete and non-terminating datalog rewriting procedure for $\mathcal{SHIQ}$. Our procedure Rewrite provides an alternative to Compile-Horn

[2]For CQ answering, our language becomes co-NP-hard w.r.t. data, whereas RL is tractable. This follows from (Lutz and Wolter 2012) already for a single axiom of type 11.

for $\mathcal{SHIQ}$. The classes of ontologies rewritable by the two procedures can be shown incomparable (e.g., Compile-Horn may not terminate on WL programs).

7 Related Work

Complexity of disjunctive datalog with negation as failure has been extensively studied (Ben-Eliyahu-Zohary and Palopoli 1997; Eiter, Gottlob, and Mannila 1997). The class of *head-cycle* free programs was studied in Ben-Eliyahu-Zohary and Palopoli; Ben-Eliyahu-Zohary, Palopoli, and Zemlyanker (1997; 2000), where it was shown that certain reasoning problems are tractable for such programs (fact entailment, however, remains intractable w.r.t. data).

Gottlob et al. (2012) investigated complexity of disjunctive TGDs and showed tractability (w.r.t. data complexity) of fact entailment for a class of linear disjunctive TGDs. Such rules allow for existential quantifiers in the head, but require single-atom bodies; thus, they are incomparable to WL rules. Artale et al. (2009) showed tractability of fact entailment w.r.t. data for DL-Lite$_{\mathsf{bool}}$ logics. This result is related to (Gottlob et al. 2012) since certain DL-Lite$_{\mathsf{bool}}$ logics can be represented as linear disjunctive TGDs. Finally, combined complexity of CQ answering for disjunctive TGDs was studied by Bourhis, Morak, and Pieris (2013).

Lutz and Wolter (2012) 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. (2013), who showed NEXPTIME-completeness of first-order and datalog rewritability of instance queries for $\mathcal{SHI}$.

The procedure in (Cuenca Grau et al. 2013), mentioned in Section 6, is used by Kaminski and Cuenca Grau (2013) to show first-order/datalog rewritability of two fragments of $\mathcal{ELU}$. Notably, both fragments yield linear programs. Finally, our unfolding-based rewriting procedure is motivated by the work of Afrati, Gergatsoulis, and Toni (2003) on linearisation of plain datalog programs by means of program transformation techniques (Tamaki and Sato 1984; Proietti and Pettorossi 1993; Gergatsoulis 1997).

8 Evaluation

Rewritability Experiments. We have evaluated whether realistic ontologies can be rewritten to WL (and hence to datalog) programs. We analysed 118 non-Horn ontologies from BioPortal, the Protégé library, and the corpus in (Gardiner, Tsarkov, and Horrocks 2006). To transform ontologies into disjunctive datalog we used KAON2 (Motik 2006).[3] KAON2 succeeded to compute disjunctive programs for 103 ontologies. On these, Rewrite succeeded in 35 cases: 8 programs were already datalog after CNF normalisation, 12 were linear, 12 were WL, and 3 required unfolding. Rewrite was limited to 1,000 unfolding steps, but all successful cases required at most 11 steps. On average, 73% of the predicates in ontologies were datalog, and so could be queried using a datalog engine (even if the disjunctive program could not be rewritten). We identified 15 RL$^{\sqcup}$ ontologies and obtained WL programs for 13 of them. For comparison, we implemented the procedure Compile-Horn in (Cuenca Grau et al. 2013), which succeeded on 18 ontologies, only one of which could not be rewritten by our approach.

	Our approach			HermiT			Pellet		
	dlog	disj	err	dlog	disj	err	dlog	disj	err
U01	<1s	8s		6s	107s		146s	172s	
U04	<1s	55s		50s	50s	2	—	—	—
U07	<1s	62s	3	107s	122s	2	—	—	—
U10	<1s	66s	5	176s	182s	2	—	—	—

Table 2: Average query answering times

Query Answering. We tested scalability of instance query answering using datalog programs obtained by our approach. For this, we used UOBM and DBpedia, which come with large datasets. UOBM (Ma et al. 2006) is a standard benchmark for which synthetic data is available (Zhou et al. 2013). We denote the dataset for k universities by Uk. We considered the RL$^{\sqcup}$ subset of UOBM (which is rewritable using Rewrite but not using Compile-Horn), and generated datasets U01, U04, U07, U10. DBpedia[4] is a realistic ontology with a large dataset from Wikipedia. Since DBpedia is Horn, we extended it with reasonable disjunctive axioms. We used RDFox as a datalog engine. Performance was measured against HermiT (Motik, Shearer, and Horrocks 2009) and Pellet (Sirin et al. 2007). We used a server with two Intel Xeon E5-2643 processors and 128GB RAM. Timeouts were 10min for one query and 30min for all queries; a limit of 100GB was allocated to each task. We ran RDFox on 16 threads. Systems were compared on individual queries, and on precomputing answers to all queries. All systems succeeded to answer all queries for U01: HermiT required 890s, Pellet 505s, and we 52s. Table 2 depicts average times for datalog and disjunctive predicates, and number of queries on which a system failed.[5] Pellet only succeeded to answer queries on U01. HermiT's performance was similar for datalog and disjunctive predicates. In our case, queries over the 130 datalog predicates in UOBM (88% of all predicates) were answered instantaneously (<1s); queries over disjunctive predicates were harder, since the rewritings expanded the dataset quadratically in some cases. Finally, due to its size, DBpedia's dataset cannot even be loaded by HermiT or Pellet. Using RDFox, our rewriting precomputed the answers for all DBpedia predicates in 48s.

9 Conclusion

We have proposed a characterisation of datalog rewritability for disjunctive datalog programs, as well as tractable fragments of disjunctive datalog. Our techniques can be applied to rewrite OWL ontologies into datalog, which enables the use of scalable datalog engines for data reasoning. Furthermore, our approach is not "all or nothing": even if an ontology cannot be rewritten, we can still answer queries over most (i.e., datalog) predicates using a datalog reasoner.

[3] We doctored the ontologies to remove constructs outside $\mathcal{SHIQ}$, and hence not supported by KAON2. The modified ontologies can be found on http://csu6325.cs.ox.ac.uk/WeakLinearity/

[4] http://dbpedia.org/About

[5] Average times do not reflect queries on which a system failed.

Acknowledgements

This work was supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI^3, and the FP7 project OPTIQUE.

References

Afrati, F.; Cosmadakis, S. S.; and Yannakakis, M. 1995. On datalog vs. polynomial time. *J. Comput. System Sci.* 51(2):177–196.

Afrati, F.; Gergatsoulis, M.; and Toni, F. 2003. Linearisability of datalog programs. *Theor. Comput. Sci.* 308(1-3):199–226.

Artale, A.; Calvanese, D.; Kontchakov, R.; and Zakharyaschev, M. 2009. The DL-Lite family and relations. *J. Artif. Intell. Res.* 36:1–69.

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

Ben-Eliyahu-Zohary, R.; Palopoli, L.; and Zemlyanker, V. 2000. More on tractable disjunctive datalog. *J. Log. Programming* 46(1-2):61–101.

Bienvenu, M.; ten Cate, B.; Lutz, C.; and Wolter, F. 2013. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. In *PODS*, 213–224. arXiv:1301.6479.

Bishop, B.; Kiryakov, A.; Ognyanoff, D.; Peikov, I.; Tashev, Z.; and Velkov, R. 2011. OWLim: A family of scalable semantic repositories. *Semantic Web J.* 2(1):33–42.

Bourhis, P.; Morak, M.; and Pieris, A. 2013. The impact of disjunction on query answering under guarded-based existential rules. In *IJCAI*.

Bry, F.; Eisinger, N.; Eiter, T.; Furche, T.; Gottlob, G.; Ley, C.; Linse, B.; Pichler, R.; and Wei, F. 2007. Foundations of rule-based query answering. In *Reasoning Web*, 1–153.

Cuenca Grau, B.; Motik, B.; Stoilos, G.; and Horrocks, I. 2013. Computing datalog rewritings beyond Horn ontologies. In *IJCAI*. arXiv:1304.1402.

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

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive datalog. *ACM Trans. Database Syst.* 22(3):364–418.

Gardiner, T.; Tsarkov, D.; and Horrocks, I. 2006. Framework for an automated comparison of description logic reasoners. In *ISWC*, 654–667.

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

Gottlob, G.; Manna, M.; Morak, M.; and Pieris, A. 2012. On the complexity of ontological reasoning under disjunctive existential rules. In *MFCS*, 1–18.

Hustadt, U.; Motik, B.; and Sattler, U. 2007. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *J. Autom. Reasoning* 39(3):351–384.

Kaminski, M., and Cuenca Grau, B. 2013. Sufficient conditions for first-order and datalog rewritability in ELU. In *DL*, 271–293.

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

Lutz, C., and Wolter, F. 2012. Non-uniform data complexity of query answering in description logics. In *KR*.

Ma, L.; Yang, Y.; Qiu, Z.; Xie, G. T.; Pan, Y.; and Liu, S. 2006. Towards a complete OWL ontology benchmark. In *ESWC*, 125–139.

Motik, B.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; Fokoue, A.; and Lutz, C. 2009. OWL 2 Web Ontology Language Profiles. *W3C Recommendation*.

Motik, B.; Nenov, Y.; Piro, R.; Horrocks, I.; and Olteanu, D. 2014. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *AAAI*.

Motik, B.; Shearer, R.; and Horrocks, I. 2009. Hypertableau Reasoning for Description Logics. *J. Artif. Intell. Res.* 36:165–228.

Motik, B. 2006. *Reasoning in Description Logics using Resolution and Deductive Databases*. Ph.D. Dissertation, Univesität Karlsruhe (TH), Karlsruhe, Germany.

Proietti, M., and Pettorossi, A. 1993. The loop absorption and the generalization strategies for the development of logic programs and partial deduction. *J. Log. Programming* 16(1):123–161.

Sirin, E.; Parsia, B.; Cuenca Grau, B.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical OWL-DL reasoner. *J. Web Sem.* 5(2):51–53.

Tamaki, H., and Sato, T. 1984. Unfold/fold transformation of logic programs. In *ICLP*, 127–138.

Wu, Z.; Eadon, G.; Das, S.; Chong, E. I.; Kolovski, V.; Annamalai, M.; and Srinivasan, J. 2008. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle. In *ICDE*, 1239–1248.

Zhou, Y.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2013. Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In *WWW*, 1569–1580.