Nonmonotonic Existential Rules for Non-Tree-Shaped Ontological Modelling ⋆

Despoina Magka, Markus Krötzsch, Ian Horrocks
{desmag,markus.kroetzsch,ian.horrocks}@cs.ox.ac.uk

Department of Computer Science, University of Oxford

1 Introduction

Description logic (DL) formalisms are extensively used for ontological modelling, e.g., in biology [16] and chemistry [19]. One such example is the *ChEBI* ontology [20] – a reference terminology adopted by various biological databases for chemical annotation [23,8,10]. Despite their wide range of expressive features, DLs are severely limited in their ability to represent structures that are not tree-shaped. This explains, e.g., why ChEBI does not capture molecular structures in its ontology, thus excluding its main content from logical reasoning. In order to overcome this deficiency, numerous rule-based extensions of DLs have been proposed that provide certain kinds of graph-based modelling, such as *description graphs* [27]. However, in order to retain decidability of description graphs several constraints are imposed, such as role separation restrictions and a cumbersome acyclicity condition, that restrict the range of structures that can be modelled and thus hinder practical usability [26]. Moreover, when performing structure-based classification a form of *closed-world assumption* is often needed to reason about the absence of structural features, e.g., to conclude that a molecule is inorganic if it does not contain carbon. Expressing completeness (closure) of finite structures in DLs is prohibitively inefficient, whereas nonmonotonic extensions of DLs remain largely unrealised in tools and applications [28].

This motivates the use of logical languages that draw upon rules enriched with non-monotonic negation for the representation and classification of objects with a complex internal structure. *Existential rules*—function-free Horn rules with existential quantifiers in rule heads—have been proposed as a new expressive ontological language [6,2], and can be viewed as a restricted kind of logic programs with function symbols. Recent works have considered nonmonotonic rule-based ontology languages using stratified negation [5,26], stable model semantics [13], and well-founded semantics [17]. If we additionally remove the stratification requirement, then the resulting language allows for the accurate modelling of complex finite structures such as those found in ChEBI.

Unfortunately, reasoning in these formalisms is computationally challenging. If negation is stratified, then all of these semantics agree, and programs have uniquely determined stable models; this is highly desirable and easy to check, but too restrictive for many applications. Moreover, even without negation, satisfiability, fact entailment,

⋆ This work was supported by the Royal Society, the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, "Optique", and the EPSRC projects ExODA, Score!, and MaSI3.

query answering, and the existence of finite models are all undecidable; and, while many non-stratified programs also have unique stable models, this property, too, is undecidable in general. As most ontologies are concerned with finite, uniquely determined structures, these problems prevent the use of such formalisms in ontological modelling.

We address this issue by presenting new conditions that are computationally feasible to check, and that identify a large class of programs having finite and unique stable models. These conditions are based on an analysis of whether one rule *relies* on another, in the sense that it might either be 'triggered' or 'inhibited' by the other rule's application. These relationships allow us to define *R-acyclicity* and *R-stratification*. Our contributions can be summarised as follows:

- We define *R-acyclic* and *R-stratified* logic programs, and show that recognising such programs is coNP-complete.
- We show that R-acyclic programs have finite stable models, and that reasoning is coN2ExpTime-complete (NP-complete for data complexity).
- We show that R-stratified programs have unique stable models, so that reasoning becomes deterministic, and that if programs are also R-acyclic, reasoning becomes 2ExpTime-complete (P-complete for data complexity).
- We extend reliances to exploit *constraints*, and show that this strictly generalises our earlier criteria. Reasoning complexities carry over, but deciding R-acyclicity and R-stratification under constraints is complete for Π_2^P.
- We conduct a case study with ChEBI, which demonstrates that our conditions do not preclude suitable modelling, that R-stratification can be exploited to allow the DLV reasoner [22] to scale to the large number of rules in our experiments, and that DLV can then be used to discover missing relationships in ChEBI.

This paper is an adapted version of [24]; omitted proofs and additional details can be found in the companion report [25].

2 Preliminaries

We consider a standard first-order language. We use the letters a, b for constants, f, g for functions, x, y, z, for variables, and t for terms. Lists of terms $\langle t_1, \ldots, t_n \rangle$ are abbreviated as $\mathbf{t}$, similarly for lists of variables $\mathbf{x}$. We treat lists as sets when order is irrelevant. A special nullary predicate symbol $\bot$ is used to denote falsity. We use $\mathsf{Pred}(\varepsilon)$, $\mathsf{Var}(\varepsilon)$, and $\mathsf{Const}(\varepsilon)$ to denote the predicates, variables, and constants, respectively, that occur in an expression ε. Atoms, i.e., formulae without operators, are written α, β, γ. When used like a formula, sets of atoms always denote the conjunction of their members. Nonmonotonic negation is denoted **not**. For a set A of atoms, we define $\mathbf{not}\, A := \{\mathbf{not}\, \alpha \mid \alpha \in A\}$. A *(nonmonotonic existential) rule* is of the form

$$r: \quad \forall \mathbf{x}.\forall \mathbf{z}.\, B^+ \wedge \mathbf{not}\, B^- \rightarrow \exists \mathbf{y}.H \tag{1}$$

where the *positive body* B^+, *negative body* B^-, and *head* H are sets (or conjunctions) of atoms without function symbols, such that $\mathsf{Var}(B^+) = \mathbf{x} \cup \mathbf{z}$, $\mathsf{Var}(B^-) \subseteq \mathbf{x} \cup \mathbf{z}$, and $\mathsf{Var}(H) \subseteq \mathbf{x} \cup \mathbf{y}$. We abbreviate r as (B^+, B^-, H). When writing rules as in (1), universal quantifiers are usually omitted. Sets of rules are called *(logic) programs*.

The *skolemisation* $\mathsf{sk}(r)$ of a rule r as in (1) is obtained by replacing each variable $y \in \mathbf{y}$ in H by a *skolem term* $f_y(\mathbf{x})$, where f_y is a fresh *skolem function symbol* of arity $|\mathbf{x}|$. Given a program P, we set $\mathsf{sk}(P) := \{\mathsf{sk}(r) \mid r \in P\}$. Assuming a fixed choice of skolem functions, sk is a bijection between rules and their skolemisations, which allows us to use the term *rule* liberally without risk of confusion. Our results refer to rules (or their skolemisations), and do not generally hold for arbitrary rules with function symbols.

A term or formula is *ground* if it contains no variables. Ground atoms are called *facts*. The *Herbrand universe* $\mathsf{HU}(P)$ of a program P is the set of all ground terms formed with constants and function symbols from $\mathsf{sk}(P)$ (using an auxiliary constant if $\mathsf{Const}(\mathsf{sk}(P)) = \emptyset$). The *grounding* $\mathsf{ground}(P)$ of P is the set of all rules that can be obtained from rules in $\mathsf{sk}(P)$ by uniformly replacing variables with terms from $\mathsf{HU}(P)$.

An *(Herbrand) interpretation* $\mathcal{M}$ is a set of facts with $\bot \notin \mathcal{M}$. Satisfaction is defined as usual: $\mathcal{M} \models B^+, \mathbf{not}\, B^-$ holds if $B^+ \subseteq \mathcal{M}$ and $B^- \cap \mathcal{M} = \emptyset$; $\mathcal{M} \models (B^+, B^-, H)$ if $\mathcal{M} \not\models B^+, \mathbf{not}\, B^-$ or $\mathcal{M} \models H$; and $\mathcal{M} \models P$ if $\mathcal{M} \models r$ for all $r \in P$. The *Gelfond-Lifschitz reduct* of P w.r.t. $\mathcal{M}$ is $\mathsf{GL}(P,\mathcal{M}) := \{(B^+, \emptyset, H) \mid (B^+, B^-, H) \in \mathsf{ground}(P) \text{ and } B^- \cap \mathcal{M} = \emptyset\}$. $\mathcal{M}$ is a *stable model* of P, written $\mathcal{M} \models_{\mathsf{SM}} P$, if $\mathcal{M} \models \mathsf{GL}(P,\mathcal{M})$ and there is no smaller model $\mathcal{M}' \subsetneq \mathcal{M}$ with $\mathcal{M}' \models \mathsf{GL}(P,\mathcal{M})$. We consider *cautious* entailment: for a program P and a fact α, $P \models \alpha$ if $\alpha \in \mathcal{M}$ for all stable models $\mathcal{M}$ of P. Consequences of programs can be computed with the T_P operator:

Definition 1. *For a program P, a set of facts F and $r \in P$ with $\mathsf{sk}(r) = (B^+, B^-, H)$, let $r(F) := \{H\theta \mid B^+\theta \subseteq F \text{ and } B^-\theta \cap F = \emptyset\}$. Moreover, let $T_P(F) := F \cup \bigcup_{r \in P} r(F)$ and define $T_P^0(F) := F$, $T_P^{i+1}(F) := T_P(T_P^i(F))$, and $T_P^\infty(F) := \bigcup_{i \geq 0} T_P^i(F)$.*

For a program P, a sequence of disjoint programs $\mathbf{P} = P_1, \ldots, P_n$ is a *stratification* of P if $P = \bigcup_{i=1}^n P_i$ and, for all programs $P_i, P_j \in \mathbf{P}$, rules $(B_1^+, B_1^-, H_1) \in P_i$ and $(B_2^+, B_2^-, H_2) \in P_j$, and every predicate $\mathsf{R} \in \mathsf{Pred}(H_1)$, we have: (i) if $\mathsf{R} \in \mathsf{Pred}(B_2^+)$ then $i \leq j$, and (ii) if $\mathsf{R} \in \mathsf{Pred}(B_2^-)$ then $i < j$; $P_1, \ldots, P_n$ are called *strata*. P is *stratified* if it has a stratification. The T_P operator can be used to characterise stable models; for stratified programs, we even obtain a deterministic computation procedure [1].

Fact 1 *Given a program P, a set of facts F, and a stable model $\mathcal{M} \models_{\mathsf{SM}} P \cup F$, we have $\mathcal{M} = T^\infty_{\mathsf{GL}(P,\mathcal{M})}(F)$. Also, if $\mathbf{P} = P_1, \ldots, P_n$ is a stratification of P, then $\mathcal{M} := T^\infty_{P_n}(\ldots T^\infty_{P_1}(F)\ldots)$ is the unique stable model of P if $\bot \notin \mathcal{M}$.*

3 Modelling with Nonmonotonic Rules

Rule-based formalisms are well suited for modelling relational structures, irrespective of whether these structures are tree-shaped or cyclic. We consider practical examples related to the modelling of chemical compounds in ChEBI. The structure of molecules can be readily represented as a logical structure. For example, the formula $M_{\mathsf{H2O}}(x,y,z) := \mathsf{o}(x) \wedge \mathsf{bond}(x,y) \wedge \mathsf{bond}(x,z) \wedge \mathsf{h}(y) \wedge \mathsf{h}(z)$ could represent a water molecule (using unidirectional bonds for simplicity). We model molecules as members of a unary predicate mol, related to their constituting atoms by the predicate hA (has

atom). The following rule represents methanol (CH_3OH), described by $M_{\mathsf{CH3OH}}(\mathbf{y})$:

$$\mathsf{methanol}(x) \rightarrow \exists \mathbf{y}.\mathsf{mol}(x) \wedge M_{\mathsf{CH3OH}}(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^{6} \mathsf{hA}(x, y_i) \tag{2}$$

Molecules can also be classified by their structure, e.g., to identify molecules that contain oxygen, or organic hydroxy molecules (those with a substructure C-O-H):

$$\mathsf{hA}(x,y) \wedge \mathsf{o}(y) \rightarrow \mathsf{hasO}(x) \tag{3}$$

$$M_{\mathsf{COH}}(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^{3} \mathsf{hA}(x, y_i) \rightarrow \mathsf{orgHydroxy}(x) \tag{4}$$

It is not hard to express syntactic identity with a predicate =, predefined in most rule engines; see [25] for details. Using **not** we can express syntactic inequality and define, e.g., molecules with exactly one carbon atom:

$$\textstyle\bigwedge_{i=1}^{2} (\mathsf{hA}(x, y_i) \wedge \mathsf{c}(y_i)) \wedge \mathbf{not}\ y_1 = y_2 \rightarrow \mathsf{multiC}(x) \tag{5}$$

$$\mathsf{mol}(x) \wedge \mathsf{hA}(x,y) \wedge \mathsf{c}(y) \wedge \mathbf{not}\ \mathsf{multiC}(x) \rightarrow \mathsf{oneC}(x) \tag{6}$$

Rules (2)–(6) and $\mathsf{methanol}(a)$ have a unique stable model (using $f_1, \ldots, f_6$ for (2)):

$$\mathcal{M}_1 := \{\mathsf{methanol}(a), \mathsf{hasO}(a), \mathsf{orgHydroxy}(a), \mathsf{oneC}(a), \mathsf{mol}(a), \mathsf{hA}(a, f_i(a))_{i=1}^{6}, \\ M_{\mathsf{CH3OH}}(f_1(a), \ldots, f_6(a))\}$$

We can thus conclude, e.g., that methanol is an organic hydroxy molecule. To obtain such inferences for organic hydroxy molecules in general, we can use another rule:

$$\mathsf{orgHydroxy}(x) \rightarrow \exists \mathbf{y}.M_{\mathsf{COH}}(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^{3} \mathsf{hA}(x, y_i) \tag{7}$$

Rules (3)–(7) and $\mathsf{orgHydroxy}(b)$ have a unique stable model (using $g_1, \ldots, g_3$ for (7)):

$$\mathcal{M}_2 := \{\mathsf{orgHydroxy}(b), \mathsf{hasO}(b), \\ \mathsf{hA}(b, g_i(b))_{i=1}^{3}, M_{\mathsf{COH}}(g_1(b), g_2(b), g_3(b))\}$$

So, organic hydroxy molecules contain oxygen, as expected. However, if we consider all of the above rules and facts together, then rather than $\mathcal{M}_1 \cup \mathcal{M}_2$ we obtain $\mathcal{M}_1 \cup \mathcal{M}_2 \cup \{\mathsf{hA}(a, g_i(a))_{i=1}^{3}, M_{\mathsf{COH}}(g_1(a), g_2(a), g_3(a)), \mathsf{multiC}(a)\} \setminus \{\mathsf{oneC}(a)\}$ as the unique stable model, since rule (7) is applicable to $\mathsf{orgHydroxy}(a)$. Thus, the stable model no longer faithfully represents methanol, which is wrongly classified as multi-carbon.

Nonmonotonic negation can be used to overcome this problem. We replace rules (4) and (7) by the following, where we abbreviate orgHydroxy by oH:

$$M_{\mathsf{COH}}(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^{3} \mathsf{hA}(x, y_i) \wedge \mathbf{not}\ n(y_i) \rightarrow \mathsf{oH}(x) \wedge r(x) \tag{8}$$

$$\mathsf{oH}(x) \wedge \mathbf{not}\ r(x) \rightarrow \exists \mathbf{y}.M_{\mathsf{COH}}(\mathbf{y}) \wedge \textstyle\bigwedge_{i=1}^{3} \mathsf{hA}(x, y_i) \wedge n(y_i) \tag{9}$$

The predicates r ('recognised') and n ('new') ensure that only one of these rules is applicable to a given structure. The above facts with rules (2), (3), (5), (6), (8), and (9) have the unique stable model $\mathcal{M}_1 \cup \mathcal{M}_2 \cup \{r(a), n(g_1(b)), n(g_2(b)), n(g_3(b))\}$, as desired. However, the resulting set of rules is not stratified, which causes various problems. First, we cannot be sure that the stable model will be unique for other sets of facts. Second, rule engines may need to apply more complex algorithms to find the stable model. Our experiments in Section 7 suggest that this may cause performance issues that prevent rule engines from computing entailments at all. The goal of this work is to overcome these issues.

4 Positive Reliances and R-Acyclicity

As recalled in Fact 1, every stable model of a logic program can be obtained from a (possibly infinite) sequence of consecutive rule applications. Insights about the semantics of a program can thus be gained by analysing, for all pairs of rules r_1 and r_2, whether applying r_1 can potentially enable applying r_2 later. We next formalise this idea of *positive reliance* between rules and define R-acyclic programs, which have finite stable models.

Definition 2 (Positive Reliance). *Let r_1 and r_2 be rules such that $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ and $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$; w.l.o.g. assume that $\mathsf{Var}(r_1) \cap \mathsf{Var}(r_2) = \emptyset$. Rule r_2* positively relies *on r_1 (written $r_1 \xrightarrow{+} r_2$) if there exists a set of facts F that contains no skolem terms and a substitution θ such that:*

$$B_1^+\theta \subseteq F \quad \text{(P1)} \qquad B_1^-\theta \cap F = \emptyset \quad \text{(P2)} \qquad B_2^+\theta \subseteq F \cup H_1\theta \quad \text{(P3)}$$

$$B_2^-\theta \cap (F \cup H_1\theta) = \emptyset \quad \text{(P4)} \qquad B_2^+\theta \not\subseteq F \quad \text{(P5)} \qquad H_2\theta \not\subseteq F \cup H_1\theta \quad \text{(P6)}$$

Thus, $r_1 \xrightarrow{+} r_2$ holds if there is a situation where r_1 is applicable (P1)/(P2), r_2 is not applicable (P5), and applying r_1 allows r_2 to derive something new (P3)/(P4)/(P6).

Example 1 *Consider rule $r_{(4)}$ of* (4) *and rule $r'_{(7)}$ obtained from* (7) *by replacing x with x'. It is $r_{(4)} \xrightarrow{+} r'_{(7)}$ since* (P1)–(P6) *are satisfied by $F := \{M_{\mathsf{COH}}(\mathbf{b})\} \cup \{\mathsf{hA}(a, b_i)\}_{i=1}^{3}$ and $\theta := \{x \mapsto a, \mathbf{y} \mapsto \mathbf{b}, x' \mapsto a\}$. But it is $r'_{(7)} \not\xrightarrow{+} r_{(4)}$. Intuitively, $r_{(4)}$ can only derive facts that are already necessary to apply $r'_{(7)}$ in the first place, thus violating* (P6).

Similar notions have been considered in the past. The *activation* relation by Greco et al. [18] is similar to Definition 2, but allows F to contain function terms to accommodate arbitrary programs with functions. Our stronger restriction is needed to show $r'_{(7)} \not\xrightarrow{+} r_{(4)}$ in Example 1. This illustrates how we can take advantage of the specific structure of existential rules to discard certain potential interactions. Other similar notions are the $\prec$ relation by Deutsch et al. [12] and the *rule dependency* by Baget et al. [2], neither of which cover negation. Baget et al. omit condition (P6), needed to show $r'_{(7)} \not\xrightarrow{+} r_{(4)}$.

If a finite program has an infinite stable model, some rule with an existential quantifier must be applicable an infinite number of times. This, however, requires that there is a cycle in rule reliances, motivating the following definition.

Definition 3 (R-Acyclic). *A program P is* R-acyclic *if there is no cycle of positive reliances $r_1 \xrightarrow{+} \ldots \xrightarrow{+} r_n \xrightarrow{+} r_1$ that involves a rule with an existential quantifier.*

Example 2 *The complete list of positive reliances for the rules $r_{(2)}, \ldots, r_{(7)}$ is $r_{(2)} \xrightarrow{+} r_{(3)}$, $r_{(2)} \xrightarrow{+} r_{(4)}$, $r_{(2)} \xrightarrow{+} r_{(5)}$, $r_{(2)} \xrightarrow{+} r_{(6)}$, $r_{(4)} \xrightarrow{+} r_{(7)}$, $r_{(7)} \xrightarrow{+} r_{(3)}$, $r_{(7)} \xrightarrow{+} r_{(5)}$, and $r_{(7)} \xrightarrow{+} r_{(6)}$. Thus the program is R-acyclic. To model $=$, we assume that $y_i = y_i$ is derived for all existential variables y_i.*

We prove that checking positive reliance for two rules is NP-complete. Similar results are shown by Deutsch et al. [12] and by Baget et al. [3] for rules without negation. The complexity refers to the size of the two involved rules rather than to the size of the whole program: in practice, positive reliances can be checked efficiently by checking the applicability of one of the rules to a linear number of facts.

Theorem 1. *Given rules r_1 and r_2, the problem of deciding whether $r_1 \xrightarrow{+} r_2$ is* NP*-complete. Checking whether a program P is R-acyclic is* coNP*-complete.*

The main result of this section shows that entailment under stable model semantics is decidable for R-acyclic programs. Hardness for coN2ExpTime is shown by reducing the word problem of 2ExpTime-bounded non-deterministic Turing machines to fact entailment, adapting constructions by Calì et al. [7] and Krötzsch and Rudolph [21].

Theorem 2. *Let P be an R-acyclic program and let $F \cup \{\alpha\}$ be a set of facts. Every stable model of $P \cup F$ has size doubly exponential in the size of P and polynomial in the size of F. Deciding $P \cup F \models \alpha$ is* coN2ExpTime*-complete w.r.t. program complexity and* coNP*-complete w.r.t. data complexity.*

5 Negative Reliances and R-Stratification

While positive reliances capture when one rule 'triggers' another, the use of nonmonotonic negation may cause the opposite interaction where one rule 'inhibits' another; we formalise this by defining *negative reliances* between rules. This suggests a new kind of *stratification*, which generalises the classical notion but can still be decided efficiently.

Definition 4 (Negative Reliance). *Let r_1 and r_2 be rules such that $\mathsf{sk}(r_1) = (B_1^+, B_1^-, H_1)$ and $\mathsf{sk}(r_2) = (B_2^+, B_2^-, H_2)$; w.l.o.g. assume that $\mathsf{Var}(r_1) \cap \mathsf{Var}(r_2) = \emptyset$. Rule r_2* negatively relies *on r_1 (written $r_1 \xrightarrow{-} r_2$) if there exists a set of facts F that contains no skolem terms and a substitution θ such that:*

$$B_1^+\theta \subseteq F \quad \text{(N1)} \qquad B_1^-\theta \cap F = \emptyset \quad \text{(N2)} \qquad B_2^+\theta \subseteq F \quad \text{(N3)} \qquad B_2^-\theta \cap H_1\theta \neq \emptyset \quad \text{(N4)} \qquad B_2^-\theta \cap F = \emptyset \quad \text{(N5)}$$

Example 3 *Consider rule $r_{(8)}$ of* (8)*, and rule $r'_{(9)}$ obtained from* (9) *by variable x with x'. We can show $r_{(8)} \xrightarrow{-} r'_{(9)}$ using $F := \{\mathsf{oH}(a), M_{\mathsf{COH}}(\mathbf{b})\} \cup \{\mathsf{hA}(a,b_i)\}_{i=1}^3$ and $\theta := \{x \mapsto a, \mathbf{y} \mapsto \mathbf{b}, x' \mapsto a\}$. Conversely, $r'_{(9)} \not\xrightarrow{-} r_{(8)}$ similarly to Example 1.*

Definition 5 (R-Stratification). *A sequence of disjoint programs $\mathbf{P} = P_1, \ldots, P_n$ is an* R-stratification *of a program P if $P = \bigcup_{i=1}^n P_i$ and, for every two programs $P_i, P_j \in \mathbf{P}$ and rules $r_1 \in P_i$ and $r_2 \in P_j$, we have:*

$$\text{if } r_1 \xrightarrow{+} r_2 \text{ then } i \leq j \qquad \text{and} \qquad \text{if } r_1 \xrightarrow{-} r_2 \text{ then } i < j.$$

P is R-stratified *if it has an R-stratification.*

Example 4 *For P consisting of rules $r_{(2)}$, $r_{(3)}$, $r_{(5)}$, $r_{(6)}$, $r_{(8)}$, and $r_{(9)}$ we obtain the reliances $r_{(2)} \xrightarrow{+} r_{(8)} \xrightarrow{-} r_{(9)} \xrightarrow{+} r_{(3)}$, $r_{(2)} \xrightarrow{+} r_{(3)}$, $r_{(2)} \xrightarrow{+} r_{(6)}$, $r_{(2)} \xrightarrow{+} r_{(5)} \xrightarrow{-} r_{(6)}$, $r_{(9)} \xrightarrow{+} r_{(5)}$, and $r_{(9)} \xrightarrow{+} r_{(6)}$. An R-stratification of P is therefore given by $P_1 := \{r_{(2)}, r_{(8)}\}$, $P_2 := \{r_{(3)}, r_{(5)}, r_{(9)}\}$, and $P_3 := \{r_{(6)}\}$. In contrast, P is not stratified due to $r_{(8)}$ and $r_{(9)}$.*

Proposition 1. *If P is stratified, then P is R-stratified.*

Example 4 and Proposition 1 above show that R-stratification properly contains stratification. The graph structure that is induced by reliances, defined next, can be used to decide R-stratification in practice, as shown in Proposition 2 below.

Definition 6 (Graph of Reliances). *For a program P, the* graph of reliances $\mathsf{GoR}(P)$ *is a directed graph that has the rules of P as its vertices and two sets of edges:* positive edges *that correspond to the positive reliances of P and* negative edges *that correspond to the negative reliances of P.*

Proposition 2. *P is R-stratified iff its graph of reliances $\mathsf{GoR}(P)$ contains no directed cycle with a negative edge.*

From the previous result it is clear that, given the graph of reliances, R-stratification can be decided in polynomial time. The overall complexity is therefore dominated by the complexity of checking individual reliances—in this sense, it is polynomial in the total number of rules, and coNP-complete only in the maximal size of a rule. Moreover, in contrast to the NP-completeness of checking positive reliances (Theorem 1), negative reliances can be detected in polynomial time.

Theorem 3. *Given rules r_1 and r_2, it can be decided in polynomial time whether $r_1 \dashrightarrow r_2$. Checking whether a program P is R-stratified is* coNP*-complete.*

It remains to show that R-stratified programs have at most one stable model, and that this model can always be obtained by repeated application of rules according to their stratification. This leads to a semi-decision procedure for entailment. If the program is also R-acyclic, we obtain a decision procedure and tight complexity bounds.

Note that Definition 4 does not include a condition that corresponds to (P6) from Definition 2. Indeed, as the next example shows, such a condition would not lead to a notion of R-stratification that ensures unique stable models.

Example 5 *Given the rules $r_1 : \mathbf{not}\ \mathsf{p} \to \mathsf{q}$ and $r_2 : \mathsf{q} \to \mathsf{p}$, we find that $r_1 \xrightarrow{+} r_2$ and $r_2 \dashrightarrow r_1$, so that the program is not R-stratified. Indeed, it has no stable models for the empty set of facts. Yet, if we required $H_2\theta \not\subseteq F$ in Definition 4, then $r_2 \dashrightarrow r_1$ would not hold, and the program would be R-stratified. Intuitively speaking, negative reliances do not just consider the case where r_2 could derive something new, but also the case where r_2 has already been used in a derivation that is no longer justified after applying r_1.*

We now define a computation scheme that can be used to obtain the unique stable model of R-stratified programs, or to derive a contradiction $\bot$ if no such model exists.

Definition 7. *For a set of facts F and a program P with R-stratification $\mathbf{P} = P_1, \ldots, P_n$, define $S^0_{\mathbf{P}}(F) := F$ and*

$$S^{i+1}_{\mathbf{P}}(F) := T^{\infty}_{P_{i+1}}(S^i_{\mathbf{P}}(F)) \quad \textit{for} \quad 0 \leq i < n.$$

For the remainder of this section, let P denote an R-stratified program with R-stratification $\mathbf{P} = P_1, \ldots, P_n$, let F denote a set of facts, and define $S^i_{\mathbf{P}} := S^i_{\mathbf{P}}(F)$.

We first show that $S^n_{\mathbf{P}}$ is a (not necessarily unique) stable model of $F \cup P$, provided that $\bot \notin S^n_{\mathbf{P}}$. The next two lemmas are key ingredients to this proof. Intuitively speaking, Lemma 1 asserts that, if the body of a rule $r \in P_i$ is satisfied at some point while computing $S^i_{\mathbf{P}}$, then it will remain satisfied in all later stages of the computation. The crucial claim is that the negative part of the rule will not be derived at any later stage. The proof of Lemma 1 relies on the definition of $\dashrightarrow$.

Lemma 1. *Consider numbers $1 \leq i \leq j \leq k \leq n$ and $\ell \geq 0$, a rule $r \in P_i$ with skolemisation $\mathsf{sk}(r) = (B^+, B^-, H)$, and a substitution θ. Then $T_{P_j}^{\ell}(S_{\mathbf{P}}^{j-1}) \models B^+\theta, \mathbf{not}\, B^-\theta$ implies $S_{\mathbf{P}}^{k} \models B^+\theta, \mathbf{not}\, B^-\theta$.*

Lemma 2 complements the previous result. Intuitively speaking, it states that a rule $r \in P_i$, which is clearly satisfied after computing $S_{\mathbf{P}}^{i}$, will remain satisfied in all later stages of the computation. The key part of this claim concerns the case that r is satisfied because its positive body is not satisfied. In this case, the positive body will never become satisfied later on, unless the head of the rule becomes satisfied as well. This argument hinges upon the definition of $\xrightarrow{+}$.

Lemma 2. *Consider numbers $1 \leq i < j \leq k \leq n$, a rule $r \in P_i$, and a substitution θ. Then $S_{\mathbf{P}}^{j} \models \mathsf{sk}(r)\theta$ implies $S_{\mathbf{P}}^{k} \models \mathsf{sk}(r)\theta$.*

Using Lemmas 1 and 2, we can show the following result.

Proposition 3. *If $\bot \notin S_{\mathbf{P}}^{n}$, then $S_{\mathbf{P}}^{n} \models_{\mathsf{SM}} F \cup P$.*

The main result of this section is that stable models of R-stratified programs are unique. We prove so by first showing that $\mathcal{M} \models_{\mathsf{SM}} P \cup F$ implies $S_{\mathbf{P}}^{n} = \mathcal{M}$, which is established by showing inductively that for all $k \in \{0, \ldots, n\}$, $S_{\mathbf{P}}^{k} = T^{\infty}_{\mathsf{GL}(\bigcup_{i=1}^{k} P_i, \mathcal{M})}(F)$.

Theorem 4. *If $\bot \notin S_{\mathbf{P}}^{n}$, then $S_{\mathbf{P}}^{n}$ is the unique stable model of $F \cup P$. Otherwise $F \cup P$ does not have a stable model.*

The complexity of reasoning for programs that are both R-acyclic and R-stratified can thus be improved. Hardness is shown by using a reduction similar to Theorem 2 but with a deterministic Turing machine, that results in an R-stratified constructed program.

Theorem 5. *Let P be an R-acyclic R-stratified program, let F be a set of facts, and let α be a fact. Deciding $P \cup F \models \alpha$ is* 2ExpTime*-complete w.r.t. program complexity and* P*-complete w.r.t. data complexity.*

6 Reliances under Constraints

To widen the classes of logic programs with unique stable models, it has been proposed to study stratification for a particular set of facts [4]. Indeed, it might be that a program that does not have a unique stable model for all sets of facts still has a unique stable model for all sets of facts that arise in the context of a given application. In this section, we therefore propose a generalisation of R-acyclicity and R-stratification that considers *constraints*, that is, rules of the form $B^+ \to \bot$ where B^+ is a set of atoms. As Example 6 suggests, constraints restrict the possible inputs so that more programs are stratified.

Example 6 *Organic molecules are those containing carbon and each inorganic entity is a molecule of geological origin:*

$$
\begin{aligned}
r_1 &: \quad \mathsf{mol}(x) \wedge \mathsf{hA}(x,y) \wedge \mathsf{c}(y) \to \mathsf{organic}(x) \\
r_2 &: \quad \mathsf{mol}(x) \wedge \mathbf{not}\ \mathsf{organic}(x) \to \mathsf{inorganic}(x) \\
r_3 &: \quad \mathsf{inorganic}(x) \to \mathsf{mol}(x) \wedge \mathsf{geoOrigin}(x)
\end{aligned}
$$

It is easily checked that $r_1 \xrightarrow{-} r_2 \xrightarrow{+} r_3 \xrightarrow{+} r_1$, so $\{r_1, r_2, r_3\}$ is not R-stratified by Proposition 2. Although the program has a unique stable model for all sets of facts, there is no stratified order of rule applications that produces the stable model. In particular, the set of facts $\{\mathsf{inorganic}(a), \mathsf{hA}(a,b), \mathsf{c}(b)\}$ requires us to apply r_3 before r_1. This situation is undesired, since inorganic molecules usually do not contain carbon, and a refined notion of reliance should take this into account.

Definition 8 (Reliances under Constraints). *Let r_1 and r_2 be rules, and let C be a set of constraints.*

- r_2 positively relies on r_1 under C *(written $r_1 \xrightarrow{+}_C r_2$) if there exists a set of facts F and a substitution θ that satisfy the conditions in Definition 2, and where $F \models C$.*
- r_2 negatively relies on r_1 under C *(written $r_1 \xrightarrow{-}_C r_2$) if there exists a set of facts F and a substitution θ that satisfy the conditions in Definition 4, and where $F \models C$.*

The classes of programs that are R-acyclic under C *and* R-stratified under C *are defined as in Definition 3 and 5, respectively, but using $\xrightarrow{\pm}_C$ instead of $\xrightarrow{\pm}$.*

It should be noted that our earlier results treat constraints like any other rule of P. This is still possible here, e.g., if some constraints are not deemed to be relevant for showing stratification. Indeed, the fewer constraints are part of C, the fewer additional checks are needed to compute reliances.

Example 7 *Consider the rules of Example 6 and the constraint $c : \mathsf{inorganic}(x) \wedge \mathsf{hA}(x,y) \wedge \mathsf{c}(y) \rightarrow \bot$. With $C := \{c\}$, we find $r_3 \not\xrightarrow{+}_C r_1$, and indeed $P_1 := \{r_1\}$, $P_2 := \{r_2, r_3\}$ is an R-stratification under these constraints.*

Adding constraints increases the complexity of testing positive reliances from NP to Σ_2^P, i.e. they can be checked in polynomial time by a nondeterministic Turing machine using an NP oracle. As before, the NP computations correspond to checking the applicability of a rule or constraint to a small set of facts, for which efficient implementations exist. A lower bound is obtained by reducing satisfiability of a quantified Boolean formula $\exists \mathbf{p}.\forall \mathbf{q}.\varphi$ to testing a positive reliance under a set of constraints.

Theorem 6. *Given rules r_1 and r_2, and a set of constraints C, deciding $r_1 \xrightarrow{+}_C r_2$ is Σ_2^P-complete. Checking if a program P is R-acyclic under constraints is Π_2^P-complete.*

As before, the relations $\xrightarrow{+}_C$ and $\xrightarrow{-}_C$ induce a graph of reliances under constraints. Analogously to Proposition 2, we can show that P is R-stratified under constraints if and only if this graph does not contain cycles that involve $\xrightarrow{-}_C$. This is the basis for deciding R-stratification under constraints, leading to the following result.

Theorem 7. *Given rules r_1 and r_2, and a set of constraints C, the problem of deciding $r_1 \xrightarrow{-}_C r_2$ is in Δ_2^P. Checking if a program P is R-stratified under C is Π_2^P-complete.*

Given an R-stratification of P under constraints C, we can again define a computation scheme to obtain unique stable models. C in this case is evaluated on all strata, though one can also defer constraint checking to the highest stratum.

Definition 9. *For a set of facts F and a program P with R-stratification $\mathbf{P} = P_1, \ldots, P_n$ under constraints C, define $S^0_{\mathbf{P},C}(F) := T_C(F)$ and*

$$S^{i+1}_{\mathbf{P},C}(F) := T^{\infty}_{P_{i+1} \cup C}(S^i_{\mathbf{P},C}(F)) \quad \textit{for} \quad 0 \leq i < n.$$

The following result can be shown using the same overall proof structure as in Section 5. The main difference is that in all arguments that discuss potential reliances between rules, we also need to show satisfaction of the constraints. This is usually a consequence of the assumption that $\bot$ is not derived.

Theorem 8. *If $\bot \notin S^n_{\mathbf{P},C}(F)$, then $S^n_{\mathbf{P},C}(F)$ is the unique stable model of $F \cup P \cup C$, or else $F \cup P \cup C$ has no stable model.*

Theorems 2 and 5 carry over for R-acyclic and R-stratified programs under constraints:

Theorem 9. *For a set of facts F, a fact α, and a program P that is R-acyclic under a set of constraints C, deciding $P \cup F \cup C \models \alpha$ is* coN2ExpTime*-complete (*coNP*-complete) w.r.t. program (data) complexity. If P is also R-stratified under C, deciding $P \cup F \cup C \models \alpha$ becomes* 2ExpTime*-complete (*P*-complete) w.r.t. program (data) complexity.*

7 Experimental Evaluation

In order to assess the practical utility of our solution, we conducted a case study with ChEBI. Our test datasets, software, and detailed results are published online [25].

The ChEBI database (release 97) contains about 20,000 molecular structures and taxonomic relations for about 8,000 chemical classes, while the DL-based ontology contains taxonomic information only. To obtain rules for reasoning, we considered a sample of 500 molecules, with sizes ranging from 2 to 138 atoms. The structure of each molecule (given in *MDL Molfile* format) was converted to rules of the form (2). Chemical classes, such as *one-carbon molecule* or *organic hydroxy*, do not have machine-readable descriptions in ChEBI. We selected 50 chemical classes and manually formalised their human-readable descriptions as rules, such as (3) and (6). In addition, we defined 30 molecule classes that are characterised by small substructures (functional groups of 2 to 8 atoms), e.g., organic hydroxy. We modelled each with two rules of the form (8) and (9), using distinct predicates r and n for each pair of rules. Finally, existential quantifiers were skolemised, and conjunctions in rule heads were decomposed into multiple rules. This led to a program P with 78,957 rules, the largest of which had 38 body atoms (8 negative). P was not stratified, but was R-stratified and R-acyclic. In addition, we generated a set F of 530 facts of the form $C(a_C)$, one for each molecule or functional group. This allowed us to compute subsumptions between chemical classes: C is subsumed by C' iff $C'(a_C)$ is in the unique stable model of $P \cup F$.

We ran experiments on a desktop computer (2GHz quad-core CPU, 4GB RAM) running Linux. In a first experiment, we tried to compute a stable model of $P \cup F$ using DLV [22], but the system failed to compute this result within a time limit of 600 seconds. In a second experiment, we split P into R-strata and consecutively computed the stable model of each stratum. Of the five R-strata of P, the first stratum P_1 contained

78,251 rules, while the 706 rules of the remaining four R-strata formed a stratified program P_2^5. We thus used DLV to compute the stable model of $P_1 \cup F$, converted the result into a new set of facts $S_{\mathbf{P}}^1$, and used DLV to compute the stable model of $S_{\mathbf{P}}^1 \cup P_2^5$. This took 17 seconds, with 13.5 seconds being used for actual reasoning in DLV.

We obtained 8,639 non-trivial subsumptions between chemical classes, which we compared to ChEBI's manually created taxonomy. This revealed several omissions in ChEBI, e.g., the fact that every organic hydroxy (ChEBI id 33822) is an organooxygen compound (ChEBI id 36963), illustrating the practical relevance of our approach.

8 Related Work

Nonmonotonic extensions for existential rules are considered by Calì et al. [5] using stratified negation, and more recently by Gottlob et al. [17] using well-founded semantics. Another approach to nonmonotonic ontological modelling are $\mathcal{FDNC}$ programs [14], which are related to DLs and inherit many of their expressivity limitations.

Local stratification generalises stratification by considering the (infinite) groundings of normal logic programs [30]. This condition is undecidable [9], but does not generalise R-stratification (see [25] for a counterexample). Further extensions along these lines led to *weak stratification* [29], *effective stratification* [4], *modular stratification* [33], and *left-to-right dynamic stratification* [34], all of which are known or suspected to be undecidable in the presence of function symbols.

Many other works study the problem of recognising negation-free programs with finite models [15,21,11]. Deutsch et al. [12] discuss how to generalise weak acyclicity by first partitioning the program into cyclic components; in the same way, reliances can be combined with any notion of acyclicity. Negation is rarely considered. *Omega-restrictedness* uses a kind of 'stratification' to ensure finiteness of stable models [35]. Magka et al. [26] define *semantic acyclicity* to ensure finite models in reasoning about structured objects but only consider stratified negation. Finally, the use of constraints on sets of facts to enhance acyclicity and stratification resembles the use of dependencies (called *extensional constraints*) on ABoxes to optimise DL query rewriting [31,32].

9 Conclusions

We showed that nonmonotonic existential rules under a stable model semantics can address complex real-world modelling problems, and presented novel conditions to ensure efficient, deterministic reasoning in these cases. Our experiments indicate that our approach can dramatically increase the performance of existing answer set programming engines, enabling them to address new, practically relevant application areas.

For future work, it is thus very promising to integrate our approach into existing rule engines, which will also allow more extensive evaluations. Section 6 suggests that cyclic or non-stratified programs could be 'repaired' by adding suitable constraints, which could inspire new tools for rule modelling. Equality theories often lead to additional reliances, whereas datatypes and numeric constraints could be exploited to discard reliances—further work is needed to study these effects.

References

1. Apt, K.R., Bol, R.N.: Logic programming and negation: A survey. J. Log. Program. 19/20, 9–71 (1994)
2. Baget, J.F., Leclère, M., Mugnier, M.L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artif. Intell. 175(9-10), 1620–1654 (2011)
3. Baget, J.F., Mugnier, M.L., Thomazo, M.: Towards farsighted dependencies for existential rules. In: RR (2011)
4. Bidoit, N., Froidevaux, C.: Negation by default and unstratifiable logic programs. Theor. Comput. Sci. 78(1), 86–112 (1991)
5. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. In: PODS. pp. 77–86. ACM (2009)
6. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: LICS (2010)
7. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: The query answering problem. Artif. Intell. 193, 87–128 (2012)
8. Chan, J., Kishore, R., Sternberg, P., Van Auken, K.: The gene ontology: enhancements for 2011. Nucleic Acids Research 40(D1), D559–D564 (2012)
9. Cholak, P., Blair, H.A.: The complexity of local stratification. Fundam. Inform. 21(4), 333–344 (1994)
10. Croft, D., O'Kelly, G., Wu, G., Haw, R., Gillespie, M., Matthews, L., Caudy, M., Garapati, P., Gopinath, G., Jassal, B., Jupe, S., Kalatskaya, I., Mahajan, S., May, B., Ndegwa, N., Schmidt, E., Shamovsky, V., Yung, C., Birney, E., Hermjakob, H., D'Eustachio, P., Stein, L.: Reactome: a database of reactions, pathways and biological processes. Nucleic Acids Research 39(Database-Issue), 691–697 (2011)
11. Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity conditions and their application to query answering in description logics. In: KR (2012)
12. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: PODS. pp. 149–158 (2008)
13. Eiter, T., Krennwallner, T., Schneider, P., Xiao, G.: Uniform evaluation of nonmonotonic DL-programs. In: FoIKS. pp. 1–22. Springer (2012)
14. Eiter, T., Simkus, M.: FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. ACM TOCL 11(2) (2010)
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
16. Gkoutos, G., Schofield, P., Hoehndorf, R.: Computational tools for comparative phenomics: the role and promise of ontologies. Mammalian Genome 23(9–10), 669–679 (2012)
17. Gottlob, G., Hernich, A., Kupke, C., Lukasiewicz, T.: Equality-friendly well-founded semantics and applications to description logics. In: AAAI (2012)
18. Greco, S., Spezzano, F., Trubitsyna, I.: On the termination of logic programs with function symbols. In: ICLP (Tech. Comm.) (2012)
19. Hastings, J., Magka, D., Batchelor, C.R., Duan, L., Stevens, R., Ennis, M., Steinbeck, C.: Structure-based classification and ontology in chemistry. J. Cheminf. 4, 8 (2012)
20. Hastings, J., de Matos, P., Dekker, A., Ennis, M., Harsha, B., Kale, N., Muthukrishnan, V., Owen, G., Turner, S., Williams, M., Steinbeck, C.: The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. Nucleic Acids Research 41(Database-Issue), 456–463 (2013)
21. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: IJCAI. pp. 963–968 (2011)

22. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM TOCL 7(3) (2006)
23. Li, C., Donizelli, M., Rodriguez, N., Dharuri, H., Endler, L., Chelliah, V., Li, L., He, E., Henry, A., Stefan, M.I., et al.: Biomodels database: An enhanced, curated and annotated resource for published quantitative kinetic models. BMC systems biology 4(1), 92 (2010)
24. Magka, D., Krötzsch, M., Horrocks, I.: Computing stable models for nonmonotonic existential rules. In: Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013). AAAI Press (2013)
25. Magka, D., Krötzsch, M., Horrocks, I.: Stable models for nonmonotonic existential rules. Tech. rep., University of Oxford (2013)
26. Magka, D., Motik, B., Horrocks, I.: Modelling structured domains using description graphs and logic programming. In: ESWC (2012)
27. Motik, B., Cuenca Grau, B., Horrocks, I., Sattler, U.: Representing ontologies using description logics, description graphs, and rules. Artif. Intell. 173(14) (2009)
28. Motik, B., Rosati, R.: Reconciling description logics and rules. J. ACM 57(5) (2010)
29. Przymusinska, H., Przymunsinski, T.C.: Weakly stratified logic programs. Fundam. Inf. 13(1), 51–65 (Mar 1990), http://dl.acm.org/citation.cfm?id=98480.98489
30. Przymusinski, T.C.: On the declarative and procedural semantics of logic programs. J. Autom. Reasoning 5(2), 167–205 (1989)
31. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over DL-Lite ontologies. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) KR. AAAI Press (2012)
32. Rosati, R.: Prexto: Query rewriting under extensional constraints in DL - Lite. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, Ó., Presutti, V. (eds.) ESWC. Lecture Notes in Computer Science, vol. 7295, pp. 360–374. Springer (2012)
33. Ross, K.A.: Modular stratification and magic sets for Datalog programs with negation. J. ACM 41(6), 1216–1266 (1994)
34. Sagonas, K.F., Swift, T., Warren, D.S.: The limits of fixed-order computation. Theor. Comput. Sci. 254, 465–499 (2001)
35. Syrjänen, T.: Omega-restricted logic programs. In: LPNMR. pp. 267–279 (2001)