Resilient Logic Programs: Answer Set Programs Challenged by Ontologies

Sanja Lukumbuzya, Magdalena Ortiz, Mantas Šimkus
Institute of Logic and Computation, TU Wien, Austria
{lukumbuzya, ortiz}@kr.tuwien.ac.at, simkus@dbai.tuwien.ac.at

Abstract

We introduce *resilient logic programs* (RLPs) that couple a non-monotonic logic program and a first-order (FO) theory or description logic (DL) ontology. Unlike previous hybrid languages, where the interaction between the program and the theory is limited to consistency or query entailment tests, in RLPs answer sets must be 'resilient' to the models of the theory, allowing non-output predicates of the program to respond differently to different models. RLPs can elegantly express $\exists\forall\exists$-QBFs, disjunctive ASP, and configuration problems under incompleteness of information. RLPs are decidable when a couple of natural assumptions are made: (i) satisfiability of FO theories in the presence of *closed predicates* is decidable, and (ii) rules are *safe* in the style of the well-known *DL-safeness*. We further show that a large fragment of such RLPs can be translated into standard (disjunctive) ASP, for which efficient implementations exist. For RLPs with theories expressed in DLs, we use a novel relaxation of safeness that safeguards rules via predicates whose extensions can be inferred to have a finite bound. We present several complexity results for the case where ontologies are written in some standard DLs.

Introduction

Rule-based languages—especially those supporting non-monotonic negation—and ontology languages like *Description Logics (DLs)* (Baader et al. 2017) offer complementary modeling and reasoning capabilities. Indeed, rule-based languages like *Answer Set Programming (ASP)* (Brewka, Eiter, and Truszczyński 2011) are tailored to provide powerful *closed-world* reasoning about *known* objects, and features like the default negation are important when modeling dynamic domains, e.g., in reasoning about actions and change. In contrast, DLs are in general syntactic variants of first-order logic and are suitable for *open-world* reasoning, especially for reasoning about *anonymous objects*, i.e., objects whose identity is unknown but whose existence is implied.

Motivated by this contrast, combining rule-based languages and DLs into *Hybrid Knowledge Bases* (HKBs) is a well-established research topic in KR&R (Rosati 2005; Eiter et al. 2008; Motik and Rosati 2010; Knorr, Alferes, and Hitzler 2011). Such hybrid languages can be divided into two classes: the *world-centric* and the *entailment-centric* approaches. The languages in (Rosati 2005; 2006; Bajraktari, Ortiz, and Šimkus 2018) are world-centric because an intended structure (i.e., an answer set) of a HKB is a *single* first-oder structure that is "acceptable" both to the rule and to the DL component of that HKB. Intuitively, in such HKBs the rules base their inferences on a given model of the DL component, rather than accessing the *knowledge* that is entailed. In other words, this means that inferences via rules must only be consistent with the DL component, which is a rather weak way of using the knowledge stored there. The entailment-centric approaches like (Eiter et al. 2008; Motik and Rosati 2010) are the other extreme: when ontological reasoning is considered, rules can base their inferences only on the logical consequences of the DL component, which means that rules have very limited access to *individual models* of the DL component.

There are many KR problems where both extremes are inadequate, since solutions must be resilient to a range of possible scenarios. For a simple (synthetic) example, assume we are given a set of nodes and we want to generate a directed graph G such that removing any single node from G will always result in a strongly connected graph. In this example, an ontology can model the possible choices of nodes to be removed. Intuitively, in order to validate our choice of edges for G, we have to make sure that every possible induced subgraph G', obtained by removing a single node from G, is strongly connected. However, the reachability relation in G' will be different for different choices of G'. This and similar examples reveal the need for a new approach that blurs the lines between the world-centric and the entailment-centric approaches. We thus study HKBs that may process different models of the input ontology in different ways, in the spirit of world-centric approaches, but the intended answer sets, which must be resilient to the different scenarios, are defined via a universal quantification over the models of the ontology, in the spirit of entailment-centric approaches.

Our contributions can be summarized as follows:

- We introduce *resilient logic programs (RLPs)*, a formalism in which a standard ASP program $\mathcal{P}$ is paired with a first-oder theory (or a DL ontology) $\mathcal{T}$ and the predicates are divided into *output*, *response*, and *open-world* predicates. The semantics is defined via a "negotiation" between $\mathcal{P}$ and $\mathcal{T}$: the two components need to agree on an *answer set* I over the output signature, so that no matter how I is extended into a model of $\mathcal{T}$ (by interpreting the open-world predicates), the program $\mathcal{P}$ can give a matching and justified interpretation to the response predicates. Both $\exists\forall\exists$-QBFs and *disjunctive* ASP (disjunctive Datalog with negation under the answer set semantics) are naturally captured by resilient programs, and in fact, the QBF reduction shows that reasoning in RLPs is Σ_3^P-hard in data complexity, setting them apart form previous hybrid languages. We also illustrate the power of RLPs for configuration problems with incomplete information.

- Inevitably, reasoning in RLPs is undecidable unless restrictions are imposed on how rules are allowed to manipulate anonymous objects. We argue that by applying some natural restrictions, including a rule safeness condition reminiscent to the well-known DL-safeness (Motik, Sattler, and Studer 2005; Rosati 2005), decidability of reasoning can be achieved. We provide a general complexity upper bound that applies to very expressive FO fragments like the *guarded negation fragment (GFNO)*.

- We further introduce a slightly more restricted fragment of RLPs, in which theories are given as sets of positive disjunctive rules and the use of default negation in front of response predicates is restricted. These restrictions cause a decrease in computational complexity of RLPs and allow us to provide a translation into disjunctive ASP, opening up a perspective for implementation.

- We then turn to RLPs where the theory is a DL ontology, and show decidability of reasoning under a significantly relaxed rule safeness condition, based on the following intuition. Assume an ontology stating that every employee of a company can take part in at most 5 projects, and that all projects have at least one employee. This can be expressed via a pair of inclusions $\mathsf{Empl} \sqsubseteq \leq 5\, \mathsf{worksFor}.\mathsf{Proj}$ and $\mathsf{Proj} \sqsubseteq \exists \mathsf{worksFor}^-.\mathsf{Empl}$. If we happen to know that the company has n fixed employees, we can infer that there can be at most $5n$ projects in any possible world. Our setting allows to interpret Empl under the *closed-world* view, and in this case the program rules are allowed to manipulate the objects in Proj as ordinary individuals, even though the set of all projects might not be known. The relaxed safeness condition uses the ontology to identify concept names whose extension is forced to be relatively small, assuming completeness of some of predicates. Such concept names are then used to safeguard program rules. This idea of safeness is novel to the best of our knowledge, and seems useful in many settings, including the kind of configuration problems we describe below.

- Finally, for the case where ontologies are written in the well-known DLs $\mathcal{ALCHOIQ}$, $\mathcal{ALCHI}$ and DL-Lite$_\mathcal{F}$, we provide algorithms and complexity results.

The full proofs omitted here due to the space restrictions are available in the extended version.

Logic Programming Preliminaries

We assume countably infinite and mutually disjoint sets $\mathbf{S}_{\text{const}}$, $\mathbf{S}_{\text{var}}$, and $\mathbf{S}_{\text{pred}}$ of *constants*, *variables*, and *predicate symbols*, respectively. Each $p \in \mathbf{S}_{\text{pred}}$ is associated with a non-negative arity, denoted by $art(p)$. A *term* is a variable or a constant, and an *atom* is an expression $p(t_1, \ldots, t_n)$, where $p \in \mathbf{S}_{\text{pred}}$ has arity n, and $t_1, \ldots, t_n$ are terms.

A *program* is a set of *rules* of the form

$$r: \quad h \leftarrow b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m$$

where $n, m \geq 0$, $h, b_1, \ldots, b_m$ are atoms, and every variable occurring in $h, b_{n+1}, \ldots b_m$ must occur in some $b_1, \ldots, b_n$. We let $head(r) = \{h\}$, $body^+(r) = \{b_1, \ldots, b_n\}$, and $body^-(r) = \{b_{n+1}, \ldots, b_m\}$. If p is an atom, the expression $not\ p$ is a *negated atom*. A *literal* is an atom or a negated atom. A rule r is *positive* if $|body^-(r)| = 0$. A program is positive if all its rules are positive. An atom, a rule, or a program is *ground* if it contains no variables. *Facts* are ground rules of the form $h \leftarrow$, where "$\leftarrow$" is often omitted. A constraint of the form '$\leftarrow \beta$' is a shorthand for '$p \leftarrow \beta, not\ p$', where p is a propositional atom not occurring elsewhere. We use $h_1 | \cdots | h_k \leftarrow \beta$ as an abbreviation for the set of rules $\{h_i \leftarrow \beta, not\ h_1, \ldots, not\ h_{i-1}, not\ h_{i+1}, \ldots, not\ h_k \mid 1 \leq i \leq k\}$. Given a set of constants $C \subseteq \mathbf{S}_{\text{const}}$, the *grounding of a rule w.r.t.* C, in symbols $ground(r, C)$, is a set of rules obtained from r by uniformly replacing variables in r by all possible elements from C. The *grounding of a program w.r.t.* C is then $ground(\mathcal{P}, C) = \bigcup_{r \in \mathcal{P}} ground(r, C)$.

An *(Herbrand) interpretation* over $\Sigma \subseteq \mathbf{S}_{\text{pred}}$ is a set of ground atoms using only predicates from Σ (if Σ is not specified we assume $\Sigma = \mathbf{S}_{\text{pred}}$). Given an interpretation I and a set $\Sigma \subseteq \mathbf{S}_{\text{pred}}$, we let $I_{|\Sigma} = \{p(\vec{u}) \mid p(\vec{u}) \in I \text{ and } p \in \Sigma\}$.

An interpretation I is a *model* of a ground positive program $\mathcal{P}$ if $body^+(r) \subseteq I$ implies $head(r) \cap I \neq \emptyset$, for each rule $r \in \mathcal{P}$. Further, I is a *minimal model* of $\mathcal{P}$ if there exists no $J \subset I$ that is a model of $\mathcal{P}$. The semantics of programs with negation is given using a program transformation due to Gelfond and Lifschitz (1988). Given an interpretation I and a program $\mathcal{P}$, we define a *reduct* of $\mathcal{P}$ w.r.t. I as $\mathcal{P}^I = \{head(r) \leftarrow body^+(r) \mid body^-(r) \cap I = \emptyset, r \in ground(\mathcal{P}, \mathbf{S}_{\text{const}})\}$. We say that I is an *answer set* of $\mathcal{P}$ if I is a minimal model of $\mathcal{P}^I$.

Resilient Logic Programs

We next present our resilient logic programs. Syntactically, they consist of an ordinary program equipped with a function-free FO theory and a partition of the signature.

We denote the predicate symbols that occur in a program $\mathcal{P}$ and a theory $\mathcal{T}$, by $\mathsf{sig}(\mathcal{P})$ and $\mathsf{sig}(\mathcal{T})$, respectively.

Definition 1 (Syntax). *A* resilient logic program *(RLP) is a tuple* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *where* $\mathcal{P}$ *is a program,* $\mathcal{T}$ *is an FO theory, and the sets* Σ_{out}, Σ_{owa}, Σ_{re} *are a partition of* $\mathsf{sig}(\mathcal{P}) \cup \mathsf{sig}(\mathcal{T})$ *with* $\Sigma_{\mathsf{re}} \cap \mathsf{sig}(\mathcal{T}) = \emptyset$. *We call* Σ_{out} *the set of* output predicates, Σ_{owa} *the set of* open predicates, *and* Σ_{re} *the set of* response predicates *of* Π. *The predicates in* $\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}$ *are called* closed predicates *of* Π.

Our semantics uses the following generalization of a reduct. Given a program $\mathcal{P}$, an interpretation I, and $\Sigma \subseteq$

$\mathbf{S}_{\text{pred}}$, the reduct $\mathcal{P}^{I,\Sigma}$ of $\mathcal{P}$ w.r.t. I and Σ is the ground positive program obtained from $ground(\mathcal{P}, \mathbf{S}_{\text{const}})$ as follows:

1. Delete every rule r that contains a literal $p(\vec{u})$ such that
   (a) $p(\vec{u}) \in body^+(r)$, $p(\vec{u}) \notin I$, and $p \in \Sigma$,
   (b) $p(\vec{u}) \in head(r)$, $p(\vec{u}) \in I$, and $p \in \Sigma$, or
   (c) $p(\vec{u}) \in body^-(r)$ and $p(\vec{u}) \in I$.
2. In the remaining rules, delete all negated atoms and all atoms $p(\vec{u})$ with $p \in \Sigma$.

This definition is inherited from *Clopen KBs* (Bajraktari, Ortiz, and Šimkus 2018), which in turn borrow the principle from *r-hybrid KBs* (Rosati 2005). Intuitively, $\mathcal{P}^{I,\Sigma}$ is the result of partially evaluating $\mathcal{P}$ according to the facts in I, interpreting the predicates in Σ as open-world. Note that in order to compute the regular reduct $\mathcal{P}^I$ we evaluate only the negated atoms in $\mathcal{P}$. In contrast, the generalized reduct requires us to additionally evaluate all atoms $p(\vec{u})$ with $p \in \Sigma$, so that the remaining program contains no predicates from Σ. Observe that if $\Sigma = \emptyset$, $\mathcal{P}^I$ coincides with $\mathcal{P}^{I,\Sigma}$.

We next define the semantics of RLPs. For convenience, we adopt the standard Herbrand semantics of FO theories.

Definition 2 (Semantics). *Let $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ be an RLP, and let I be an interpretation over Σ_{out}. Then I is an* answer set *of Π if*

(i) there exists some model J of $\mathcal{T}$ such that $I = J_{|\Sigma_{\mathsf{out}}}$, and
(ii) for each model J of $\mathcal{T}$ with $I = J_{|\Sigma_{\mathsf{out}}}$, there is an interpretation H such that $J_{|\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}} = H_{|\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}}$ and $H_{|\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}}$ is a minimal model of $\mathcal{P}^{H,\Sigma_{\mathsf{owa}}}$.

We call H a response *to J* w.r.t. *I and Π.*

Intuitively, an answer set of an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ is an interpretation I over the output predicates that fulfills the following two conditions: (i) I is consistent with the theory, i.e., we can extend I into a model of $\mathcal{T}$ by interpreting the open predicates and (ii) no matter how we extend I into a model of $\mathcal{T}$, we can always find a matching interpretation for the response predicates that, together with I, will be justified by the program $\mathcal{P}$.

RLPs provide an easy way of modeling and solving problems with an underlying exist-forall-exist structure. More precisely, RLPs are suitable for problems where we have control over some parameters that might, to some extent, influence the environment that we can otherwise not control and is unknown to us a priori, but to which we have to be able to respond adequately. Note that different states of the environment will likely require different responses. In such scenarios, we use the theory to describe the possible states of the environment, and the rules of the program to process a given state. The partitioning of the predicates into three different sets can be intuitively explained as follows. The output predicates are the predicates whose extensions can be controlled and that do not depend on the unknown environment, but rather might influence the set of possible states of the environment one has to consider. The predicates whose extensions we have absolutely no control over are the open predicates. These predicates are used to describe the unknown parts of the environment that we must react to (e.g., things related to user input). Finally, the response predicates are the predicates used in the rules for computing responses to the environment and their extensions are dependent on the specific extensions of the open predicates.

We next illustrate RLPs through a few examples.

Example 1. *We show how to express the graph problem from the introduction as an RLP. Given nodes $n_1, \ldots, n_k$, let $\Pi = (\mathcal{P}, \mathcal{T}, \{V, E\}, \{in, out\}, \{\overline{E}, R\})$, where*

$$\begin{aligned}
\mathcal{T} = \{&\exists x out(x), \quad \forall x(V(x) \rightarrow in(x) \vee out(x)),\\
&\forall x(V(x) \rightarrow \neg in(x) \vee \neg out(x)),\\
&\forall x \forall y out(x) \wedge out(y) \rightarrow x = y\}\\
\mathcal{P} = \{&V(n_1), \cdots V(n_k), \quad E(x,y)|\overline{E}(x,y) \leftarrow V(x), V(y),\\
&R(x,z) \leftarrow R(x,y), R(y,z),\\
&R(x,y) \leftarrow E(x,y), not\ out(x), not\ out(y),\\
&\leftarrow V(x), V(y), x \neq y, not\ out(x), not\ out(y), not\ R(x,y)\}
\end{aligned}$$

In each answer set of Π, E defines the edge relation of a directed graph G such that removing any single node n_i, $i \in \{1, \ldots, k\}$, from G (i.e., n_i is the only node in out), results in a graph that is still strongly connected. For example, for $k = 4$ we have that $I = \{V(n_1), V(n_2), V(n_3), V(n_4), E(n_1,n_2), E(n_2,n_3), E(n_3,n_4), E(n_4,n_1), E(n_1,n_3), E(n_3,n_1), E(n_2,n_4), E(n_4,n_2)\}$ is an intended answer set as removing any single node from this graph (and all edges relating to this node) yields a strongly connected graph. The set obtained from I by removing, e.g., $E(n_1,n_2)$ is not an intended answer set. In such a graph, removing n_3 results in n_2 not being reachable from n_1.

We also show that RLPs can elegantly capture $\exists\forall\exists$-quantified Boolean formulas (QBFs).

Example 2. *Consider the evaluation problem for QBFs of the form $\Phi = \exists X_1, \ldots, X_n \forall Y_1, \ldots, Y_m, \exists Z_1, \ldots, Z_k \varphi.$, where φ is in 3-CNF. We define an RLP Π whose answer sets directly correspond to the truth-value assignments for $X_1, , \ldots, X_n$ for which Φ evaluates to true.*

Note that Π reflects the quantifier alternation in Φ: we want to output an assignment for the X_i (i.e., an interpretation over T_X and F_X) such that for every assignment for the open-world Y_i we can respond with an assignment for the Z_i that satisfies the constraints, i.e., the clauses in Φ. We let $\Pi = (\mathcal{P}, \mathcal{T}, \{V_X, V_Y, V_Z, T_X, F_X\}, \{T_Y, F_Y\}, \{T_Z, F_Z\})$,

$$\begin{aligned}
\mathcal{T} = \{&\forall x(V_Y(x) \rightarrow T_Y(x) \vee F_Y(x)),\\
&\forall x(V_Y(x) \wedge T_Y(x) \wedge F_Y(x) \rightarrow \bot)\}\\
\mathcal{P} = \{&V_\alpha(c_1^\alpha), \quad \ldots \quad V_\alpha(c_{n_\alpha}^\alpha),\\
&T_X(x)|F_X(x) \leftarrow V_X(x), \quad T_Z(x)|F_Z(x) \leftarrow V_Z(x),\\
&\leftarrow \sigma(L_{i,1}), \sigma(L_{i,2}), \sigma(L_{i,3}), \textit{for each clause } C_i \textit{ in } \varphi\}
\end{aligned}$$

where, given a literal l, $\sigma(l)$ is defined as:

$$\sigma(l) = \begin{cases} T_\alpha(c_i^\alpha) & \textit{if } l = \neg\alpha_i, i = 1, \ldots, n_\alpha \\ F_\alpha(c_i^\alpha) & \textit{if } l = \alpha_i, i = 1, \ldots, n_\alpha \end{cases}$$

where $\alpha \in \{X, Y, Z\}$, $n_X = n, n_Y = m$, and $n_Z = k$.

Example 3. *Assume a company has to process a fixed amount of customer orders per day. The company does not* *know what the exact configuration of these orders will be, but it knows that each of them consists of up to 5 tasks and each task requires one service offered by the company. The company has a task of selecting which services to offer so that no matter what the actual configuration of the orders is, the tasks can be scheduled to employees in a way that each task will be completed by the end of the day.*

This problem is solved by an RLP in which the offered services are captured by the output predicates, models of the theory correspond to possible configurations of orders, and the models of the program define viable schedules of tasks to employees. The answer sets of such an RLP then correspond to sets of services that, if offered, guarantee that for every configuration of orders there exists a schedule in which each task is completed by the end of the workday.

We define a program $\mathcal{P}_1$ consisting of the rules that model the timeline of the workday:

$$Next(i, i+1),\ for\ 0 \leq i < t_{max},$$
$$Time(y) \leftarrow Next(x,y), \qquad Time(x) \leftarrow Next(x,y),$$
$$ltHour(x_0, x_n) \leftarrow Next(x_0, x_1), \ldots, Next(x_{n-1}, x_n),\ 0 \leq n < 60$$

Assume that the employees work eight hours per day and the granularity of Next is one minute. We set $t_{max} = 480$.

As facts, we store the employees, the services, as well as which employee can provide which service. For simplicity assume that each service takes the same amount of time to be completed, e.g., 60 minutes, and that the company needs to process two orders per day. We also encode this information using facts. Consider, for demonstration purposes, the following set of facts:

$$\mathcal{P}_2 = \{Service(s_1), Service(s_2), Service(s_3), Employee(e_1), Employee(e_2), Order(o_1), Order(o_2), Provides(e_1, s_1), Provides(e_1, s_2), Provides(e_2, s_1), Provides(e_2, s_2), Provides(e_2, s_3)\}$$

We further introduce two binary predicates hasTask and Req for specifying which orders have which tasks associated to them and which tasks require which offered services, respectively. Assume we have an FO theory $\mathcal{T}$ expressing the following information: (i) each order has at least one and at most five tasks associated to it, (ii) each task is associated to exactly one order and (iii) each task requires exactly one offered service. We show later that such a theory can be elegantly expressed using description logics.

The rules that select services are defined as follows:

$$\mathcal{P}_3 = \{OfferedService(x) | \overline{OfferedService}(x) \leftarrow Service(x)\}$$

The set of rules $\mathcal{P}_4$ in Table 1 generates a viable schedule (Sched) consisting of tuples (x, y, z), assigning task y to employee x to be performed starting at time point z.

Let $\Sigma_{\mathsf{out}} = \{Order, OfferedService\}$, $\Sigma_{\mathsf{owa}} = \mathsf{sig}(\mathcal{T}) \setminus \Sigma_{\mathsf{out}}$, and $\Sigma_{\mathsf{re}} = \mathsf{sig}(\mathcal{P}) \setminus (\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}})$. The answer sets of the RLP $(\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, where $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3 \cup \mathcal{P}_4$, represent sets of services that can be offered and completed within the given time constraints regardless of the type of received orders. One example of such an answer set is $I = \{Order(o_1), Order(o_2), OfferedService(s_1) OfferedService(s_2)\}$. It can be verified that whatever the configurations o_1 and o_2 might be, we can always find a schedule in which all tasks are completed on time.

Note that, unlike traditional ASP, RLPs can have comparable answer sets, e.g., $\{OfferedService(s_1) Order(o_1), Order(o_2)\} \subset I$ is also an answer set of Π.

Let $J = \{OfferedService(s_3), Order(o_1), Order(o_2)\}$ and consider a model of $\mathcal{T}$ in which each order consist of five tasks associated with the service s_3. Since only employee e_2 can perform s_3, she needs to perform this service ten times. However, this takes more than the 480 minutes and so no valid schedule can be found. This means that J is not an answer set of Π.

From Disjunctive Programs to RLPs. There is a strong connection between RLPs and *disjunctive logic programs with negation under the answer set semantics* (Eiter, Gottlob, and Mannila 1997). Disjunctive logic programs consist of rules that are similar to the rules defined in the previous section, but allow disjunctions of atoms in rule heads. Namely, disjunctive rules are of the form

$$h_1 \vee \ldots \vee h_l \leftarrow b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m,$$

where $l > 1, n, m \geq 0, h_1, \ldots, h_l, b_1, \ldots, b_m$ are atoms and every variable occurring in $h_1, \ldots, h_l, b_{n+1}, \ldots, b_m$ must occur in some $b_1, \ldots, b_n$. The semantics to these programs is given analogously to the semantics of programs introduced in the preliminaries to this paper. We next show that disjunctive logic programs under the answer set semantics are fully captured by RLPs.

Theorem 1. *Every disjunctive logic program $\mathcal{P}$ can be translated in polynomial time into a resilient logic program $\Pi = (\mathcal{P}', \{\top\}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ whose answer sets coincide with those of $\mathcal{P}$.*

Proof sketch. The core idea of the translation is to use two copies of the signature of $\mathcal{P}$, both encoding possible interpretations over the signature and the constants of $\mathcal{P}$. As the theory component plays no role in this translation we use $\top$. Our strategy is the following. We first define rules that generate a candidate answer set I of $\mathcal{P}$ and use constraints to ensure that I satisfies all the rules of $\mathcal{P}$. For this we use the signature of $\mathcal{P}$ – our set of output predicates, and one of its copies – the response predicates. The other copy constitutes the set of open predicates and is used to verify that I is indeed an answer set of $\mathcal{P}$. To do this, we need to make sure that there is no interpretation $J \subsetneq I$ that is a model of $\mathcal{P}^I$. In other words, we need to check that *for all possible interpretations* J, either (i) $J = I$, (ii) there is an atom h such that $h \in J$ and $h \notin I$, i.e., $J \not\subset I$, or (iii) J violates some rule in $\mathcal{P}^I$. We define the rules, using some additional response predicates, to check for a given J if one of these conditions holds. Since we have a universal quantification over interpretations, we use open predicates to encode J. Our semantics then ensures that if I is an answer set of Π then for any possible interpretation J at least one of (i-iii) holds. Hence,

$$\begin{array}{rcl}
\mathcal{P}_4 = \{Sched(x,y,z) & \leftarrow & Task(y), Time(z), Req(y,u), Provides(x,u), not\ Illegal(x,y,z) \\
Illegal(x,y,z) & \leftarrow & Task(y), Time(z), ltHour(z, t_{max}), Employee(x) \\
Illegal(x,y,z) & \leftarrow & Sched(x',y,z'), Time(z), Employee(x), x \neq x' \\
Illegal(x,y,z) & \leftarrow & Sched(x',y,z'), Time(z), Employee(x), z \neq z' \\
Illegal(x,y,z) & \leftarrow & Task(y), Sched(x,y',z'), ltHour(z',z), z \neq z' \\
Illegal(x,y,z) & \leftarrow & Task(y), Sched(x,y',z'), Time(z), z = z', y \neq y' \\
OKTask(y) & \leftarrow & Sched(x,y,z) \qquad \leftarrow not\ OKTask(y), Task(y)\}
\end{array}$$

Table 1: Rules that check whether viable schedules exist

the answer sets of $\mathcal{P}$ and Π coincide. The exact construction can be found in the extended version. □

Decidable RLPs

The main reasoning task for RLPs is deciding the *answer set existence*, i.e., given an RLP Π, decide whether there exist an answer set I of Π. We note that other common reasoning problems like *skeptical* (resp., *brave*) *entailment* can be easily reduced to checking non-existence (resp., existence) of an answer set. In particular, a ground atom $p(\vec{c})$ is true in all answer sets of Π (i.e. $p(\vec{c})$ is a skeptical consequence) iff the result of adding $\leftarrow p(\vec{c})$ to (the program component of) Π does not have an answer set. Similarly, a ground atom $p(\vec{c})$ is true in some answer set of Π (i.e. $p(\vec{c})$ is a brave consequence) iff Π augmented with $\leftarrow$ *not* $p(\vec{c})$ does have an answer set. With this in mind, the rest of the paper is focused on the problem of deciding answer set existence.

Unsurprisingly, this task is undecidable for RLPs in their general form. This is not hard to show adapting analogous results for some well-known hybrid languages (Levy and Rousset 1998; Rosati 2007). In this section, we identify a class of RLPs for which decidability can be regained.

The first step is to introduce a *safeness* condition for RLPs that ensures that variables in the program range only over a finite number of constants. To this end, we use the notion of safeness presented in (Bajraktari, Ortiz, and Šimkus 2018) which was inspired by the well-known *DL-safeness* (Motik, Sattler, and Studer 2005; Rosati 2005):

Definition 3. *An RLP* $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *is* safe *if for each rule* $r \in \mathcal{P}$*, each variable in* r *occurs in some* $p(\vec{u}) \in body^+(r)$*, where* $p \in \Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}$.

Intuitively, for RLPs that fulfill this condition, variables in the program $\mathcal{P}$ range only over the set $\mathsf{adom}(\mathcal{P})$ of constants explicitly mentioned in $\mathcal{P}$. This is exploited to devise a terminating algorithm for answer set existence of RLPs.

An obvious further requirement for decidability is that the theory component of RLPs must belong to a fragment of FO in which satisfiability is decidable. However, we need something stronger: the theory must be in a logic for which the problem of *satisfiability under closed predicates* is decidable, i.e., given a theory $\mathcal{T}$, a set of predicates Σ regarded as closed, and an interpretation I, we can decide whether there is a model J of $\mathcal{T}$ s.t. $J_{|\Sigma} = I$.

Theorem 2. *The problem of answer set existence is decidable for safe RLPs whose theory is in a logic for which satisfiability under closed predicates is decidable.*

Proof sketch. Algorithm 1 decides the answer set existence for such RLPs. In a nutshell, the algorithm takes an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, guesses an interpretation I and subsequently uses two oracle calls to check whether I is an answer set of Π. The first call checks whether there is a model J of $\mathcal{T}$ s.t. $J_{|\Sigma_{\mathsf{out}}} = I$. The second call tries to verify whether each such model of $\mathcal{T}$ has a response w.r.t. I and Π by attempting to guess a model J' of $\mathcal{T}$ for which this does not hold. In order to check that J' has no response, another oracle call is made which tries to guess a response. If this fails, there is a counter example to I being an answer set of Π. Note that, due to our safeness condition, it suffices to make the guesses only over $\mathsf{adom}(\mathcal{P})$. Further, the subroutine isConsistent($\mathcal{T}, I, \Sigma$) checks whether there exists a model J of $\mathcal{T}$ s.t. $J_{|\Sigma} = I$. For RLPs that fulfill the conditions from Theorem 2 this problem is decidable and the subroutine terminates, yielding a decision procedure. □

A naive analysis of Algorithm 1 yields the following upper bound in terms of computational complexity:

Theorem 3. *Let* $\mathcal{L}$ *be a logic in which satisfiability under closed predicates is in some complexity class* C *that contains* NP. *Deciding answer set existence of safe RLPs with theories in* $\mathcal{L}$ *is in* $\text{NExpTime}^{\text{C}^{\text{NP}}}$.

One of the most expressive decidable fragments of FO is the so-called *guarded negation fragment (GNFO)* (Bárány, Cate, and Segoufin 2015), which extends the guarded fragment of FO and also captures most decscription logics. As in GNFO satisfiability under closed predicates is decidable in 2ExpTime (Benedikt et al. 2016; Bajraktari, Ortiz, and Šimkus 2018), by Theorem 1, answer set existence for safe RLPs with GNFO theories is decidable and belongs to the class $\text{NExpTime}^{\text{2ExpTime}^{\text{NP}}} = \text{NExpTime}^{\text{2ExpTime}}$.

Translation into Disjunctive Datalog

We now identify a fragment of RLPs that can be translated into disjunctive Datalog. First, we require that all RLPs in this fragment are safe, as described in the previous section.

We next disallow default negation in front of response predicates in rules other than constraints. This restriction has as an effect that, given a candidate answer set I and a model of the theory J that agrees with I on the output predicates, deciding whether there is a response to J can be done deterministically, which reduces the computational complexity of reasoning with such RLPs. We note that many problems, like Example 1 and Example 3, can be captured in this fragment.

Algorithm 1: Answer set existence of RLPs.

```
Algorithm hasAnswerSet($\Pi$)
    Input   : RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$
    Output: true iff $\Pi$ has an answer
    Guess an interpretation $I$ over $\Sigma_{\mathsf{out}}$ and $\mathsf{adom}(\mathcal{P})$
    return isConsistent($\mathcal{T}, I, \Sigma_{\mathsf{out}}$) and
      not hasCE($I, \Pi, \mathsf{adom}(\mathcal{P}), \Sigma_{\mathsf{out}}$)
Subroutine hasCE($I, \Pi, \Delta, \mathcal{B}$)
    Input   : Interpretation $I$, RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}},$
              $\Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$, $\Delta \subseteq \mathbf{S}_{\text{const}}$, and $\mathcal{B} \subseteq \mathsf{sig}(\mathcal{T})$
    Output: true iff there is a counter example to $I$
              being an answer set of $\Pi$
    Guess an interpretation $J$ over $\mathsf{sig}(\mathcal{P}) \cap \mathsf{sig}(\mathcal{T}) \cup$
    $\Sigma_{\mathsf{out}}$ and the constants from $\Delta$ s.t. $J_{|\Sigma_{\mathsf{out}}} = I_{|\Sigma_{\mathsf{out}}}$
    $J' \leftarrow J \cup \{\neg p(\vec{c}) \mid \vec{c} \in \Delta^{art(p)},$
        $p \in \mathsf{sig}(\mathcal{P}) \cap (\mathsf{sig}(\mathcal{T}) \setminus \mathcal{B}),$ and $p(\vec{c}) \notin J\}$
    return isConsistent($\mathcal{T}, J', \mathcal{B}$) and
        not hasResp($J, \Pi$)
Subroutine hasResp($J, \Pi$)
    Input   : Interpretation $J$,
              RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$
    Output: true iff there exists a response to $J$ w.r.t.
              $J_{|\Sigma_{\mathsf{out}}}$ and $\Pi$
    Guess an interpretation $H$ over $\mathsf{sig}(\mathcal{P})$ and the
    constants from $J$ and $\mathsf{adom}(\mathcal{P})$ s.t. $H_{|\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{owa}}} = J$
    return $H_{|\Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}}}$ is a min. model of $\mathcal{P}^{H, \Sigma_{\mathsf{owa}}}$
```

Finally, we restrict the theory component of RLPs in this fragment to essentially a set of positive disjunctive rules which allows us to easily encode formulas of the theory as rules of the program component. More formally, we consider FO formulas of the form $\forall x_1, \dots, x_l \varphi$, where φ is a disjunction of literals using only constants and variables $x_1, \dots, x_l$ in which every variable occurring in a positive literal involving an output predicate also occurs in a negative literal involving a an output predicate. This fragment of FO is particularly important for DL research as the formulas in it can be seen as positive disjunctive rules and many standard DLs are rewritable into positive disjunctive Datalog (see, e.g., (Hustadt, Motik, and Sattler 2007; Bienvenu et al. 2014)).

We next show that the considered fragment of RLPs is captured by disjunctive logic programs with negation under the answer set semantics.

Theorem 4. *Every RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ in the fragment described above can be translated into a disjunctive program whose answer sets restricted to Σ_{out} coincide with the answer sets of Π.*

Proof sketch. For this translation, we employ the technique from (Eiter and Polleres 2003) that transforms two non-disjunctive programs $\mathcal{P}_1$ and $\mathcal{P}_2$ into a single disjunctive program $\mathcal{P}_3$ whose answer sets correspond to the answer sets S of $\mathcal{P}_1$ for which $\mathcal{P}_2 \cup S$ does not have an answer set.

As the first step, we can obtain from an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ two programs $\mathcal{P}_{guess}$ and $\mathcal{P}_{check}$ that do the following:

- $\mathcal{P}_{guess}$ guesses a structure over the output predicates and makes sure that it can be extended into a model of $\mathcal{T}$ by interpreting the open predicates. Answer sets of $\mathcal{P}_{guess}$ consist of such structures, extended into models of $\mathcal{T}$.
- $\mathcal{P}_{check}$ checks, given an answer set S of $\mathcal{P}_{guess}$, whether there is a model of $\mathcal{T}$ that agrees with S on the output predicates and has no response. Such models constitute the answer sets of $\mathcal{P}_{check} \cup S$.

The second step is to use the above-mentioned technique to obtain a disjunctive program $\mathcal{P}_{solve}$ whose answer sets correspond to the answer sets S of $\mathcal{P}_{guess}$ for which $\mathcal{P}_{check} \cup S$ does not have an answer set. This results in $\mathcal{P}_{solve}$ having answer sets that correspond to structures S over output predicates (extended into models of $\mathcal{T}$ in possibly multiple different ways) for which there is no model of $\mathcal{T}$ that agrees with S on the output predicates and has no response. Thus, we have that the answer sets of $\mathcal{P}_{solve}$ (restricted to the output predicates) and the answer sets of Π coincide. $\square$

Resilient Logic Programs with DL Theories

In this section, our focus is on RLPs whose theory is specified using *description logics*.

Description Logics Preliminaries. Most description logics, in particular those considered in this paper, are fragments of FO with only unary and binary predicate symbols, and a slightly modified syntax that allows us to write formulas more concisely. We consider the following three DLs: the expressive $\mathcal{ALCHI}$ and $\mathcal{ALCHOIQ}$, and the lightweight $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$. We assume countably infinite sets $\mathbf{S}_{\text{cn}} \subseteq \mathbf{S}_{\text{pred}}$ and $\mathbf{S}_{\text{rn}} \subseteq \mathbf{S}_{\text{pred}}$ of unary predicate symbols called *concept names* and binary predicate symbols called *role names*, respectively. A *role* R is defined according to the following syntax: $R := P \mid P^-$, where $P \in \mathbf{S}_{\text{rn}}$ and P^- is called the inverse of P. We sometimes abuse the notation and write R^- to denote P if $R = P^-$. We summarize the syntax of (*complex*) *concepts* for our considered logics:

$$\begin{aligned}
\mathsf{DL\text{-}Lite}_{\mathcal{F}}:&\ C := \bot \mid A \mid \exists R\\
\mathcal{ALCHI}:&\ C := \top \mid A \mid \neg C \mid C \sqcap C \mid \exists R.C \mid \forall R.C\\
\mathcal{ALCHOIQ}:&\ C := \top \mid A \mid \neg C \mid C \sqcap C \mid \geq nR.C \mid \leq nR.C \mid \{a\},
\end{aligned}$$

where $A \in \mathbf{S}_{\text{cn}}$, R is a role, $a \in \mathbf{S}_{\text{const}}$, and $n \geq 0$,

Hereinafter, C, D denote complex concepts and R, S denote roles in the considered logic. Moreover, we abbreviate $\leq nR.C \sqcap \geq nR.C$ by $= nR.C$.

DL knowledge bases consist of two components: an *ABox* and a *TBox*. An ABox is a finite set of *assertions* of the form $C(a)$ and $R(a, b)$, where $a, b \in \mathbf{S}_{\text{const}}$. In $\mathcal{ALCHI}$ and $\mathcal{ALCHOIQ}$, a TBox is a finite set of expressions of the form $C \sqsubseteq D$, called *concept inclusions* and $R \sqsubseteq S$, called *role inclusions*. In $\mathsf{DL\text{-}Lite}_{\mathcal{F}}$, a TBox consists of concept inclusions of the form $C \sqsubseteq D$ and $C \sqsubseteq \neg D$ and axioms of the form (funct R). We can view TBoxes as FO theories (see (Borgida 1996)). For example, the FO theory in Example 3

	$\mathcal{ALCHI}$	DL-Lite$_\mathcal{F}$	$\mathcal{ALCHOIQ}$
combined complexity	NExpTime$^{\text{NP}}$- c	NExpTime$^{\text{NP}}$- c	in NExpTime$^{\text{NExpTime}}$
combined complexity with bounded predicate arities	ExpTime - c	Σ_3^P- c	in NP$^{\text{NExpTime}}$
data complexity	Σ_3^P- c	Σ_3^P- c	in NP$^{\text{NExpTime}}$

Table 2: Complexity of answer set existence for RLPs (c denotes completeness results).

corresponds to the following $\mathcal{ALCHOIQ}$ TBox $\mathcal{T}$:

$$Order \sqsubseteq \geq 1 hasTask.\top \sqcap \leq 5 hasTask.\top,$$

$$\geq 1 Req^{-}.\top \sqsubseteq OfferedService,\ Task \sqsubseteq = 1 Req.\top,$$

$$Task \sqsubseteq = 1 hasTask^{-}.Order,\ \geq 1 hasTask^{-}.\top \sqsubseteq Task$$

We say that an interpretation is a model of a TBox if it is a model of its corresponding FO theory.

Algorithm 1 and the known complexity results on ASP and DLs with closed predicates allows us to characterize the complexity of safe RLPs with DL theories. We provide results for both combined complexity, measured in terms of the size of the whole RLP, as well as for data complexity in which only the size of the facts matter, while the size of the theory and the non-factual rules of the RLP's program component are treated as constant. Additionally, as the program component can contain predicates of arbitrary arities, we also provide some results for the case where these arities are bounded by a constant.

Theorem 5. *Table 2 provides correct complexity results for the deciding the answer set existence for safe RLPs.*

Note that Σ_p^3-hardness in data-complexity of $\mathcal{ALCHI}$ and DL-Lite$_\mathcal{F}$ can be shown by slightly modifying the reduction from $\exists\forall\exists$-QBFs to RLPs in Example 2. We discuss this and other results in the extended version.

Relaxed Safeness. Recall the RLP from Example 3. This RLP is unsafe as there are rules that contain atoms of the form $Task(x)$ in which the variable x is not safeguarded by a closed predicate. However, as we know the number n of orders that need to be processed per day, and the theory stipulates that each order has at most five tasks and each task belongs to exactly one order, we conclude that the maximum number of tasks to perform is $5n$. This means that even though the set of tasks is not known, there is an upper bound on the number of constants that the variable x in a positive atom of the form $Task(x)$ can range over. Thus, it is safe to consider $Task$ as a quasi-closed predicate and allow positive literals using $Task$ to safeguard variables in rules.

We next formalize this intuition and significantly relax our previous notion of safeness. To this end, we introduce the notion of *bounded concept name*.

Definition 4. *Let $\mathcal{T}$ be an $\mathcal{ALCHOIQ}$ TBox, Σ be a finite set with $\Sigma \subseteq \boldsymbol{S}_{pred}$, and $\mathcal{N} = \{\{c\} \mid c$ occurs in $\mathcal{T}\}$. The set $\boldsymbol{B}_{cn}(\mathcal{T}, \Sigma)$ of* bounded concept names *in $\mathcal{T}$ w.r.t. Σ is the smallest set that contains every concept name $A \in \mathsf{sig}(\mathcal{T})$ for which at least one of the following holds:*

1. $A \in \Sigma$,
2. there is a role R in $\mathsf{sig}(\mathcal{T}) \cap \Sigma$ s.t. $\mathcal{T} \models A \sqsubseteq \exists R.\top$,
3.$\mathcal{T} \models A \sqsubseteq \bigsqcup_{B \in \mathcal{B} \cup \mathcal{N}} B$, or
4. there exists $B \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma) \cup \mathcal{N}$, a role R occurring in $\mathcal{T}$, and integers $n, m \geq 0$ s.t. $\mathcal{T} \models A \sqsubseteq \geq mR.B$ and $\mathcal{T} \models B \sqsubseteq \leq nR^{-}.A$.

If $\mathcal{T}$ is in $\mathcal{ALCHI}$, item 4. in the definition above is omitted and $\mathcal{N} = \emptyset$. Similarly, if $\mathcal{T}$ is in DL-Lite$_\mathcal{F}$, item 3. is omitted and item 4. is replaced by the following:

4a. there exists $B \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma)$ and a role R occurring in $\mathcal{T}$ s.t. $\mathcal{T} \models A \sqsubseteq \exists R^{-}$, $\mathcal{T} \models \exists R \sqsubseteq B$, and (funct R) $\in \mathcal{T}$.

Note that, given $\mathcal{T}$ and Σ, computing the set $\mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma)$ requires a polynomial number of steps, some of which use an entailment test as an oracle.

Proposition 1. *Let $\mathcal{T}$ be a TBox in one of the DLs above, $\Sigma \subseteq \boldsymbol{S}_{pred}$, and $b \geq 0$. We can compute a constant b' s.t. $|\{c \mid A(c) \in J, A \in \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma)\}| \leq b'$ holds in every model J of $\mathcal{T}$ in which $|\{c \mid p(\vec{c}) \in J, p \in \Sigma, c$ occurs in $\vec{c}\}| \leq b$. If $\mathcal{T}$ is in $\mathcal{ALCHOIQ}$, b' is at most exponential in the size of $\mathcal{T}$. Otherwise, b' is polynomial in the size of $\mathcal{T}$.*

We remark that the definition above is incomplete, in the sense that $\mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma)$ need not contain all concept names for which a bound exists. Nonetheless, it is sufficient for us to relax our previous notion of safeness. Given a program $\mathcal{P}$, for each $p \in \mathsf{sig}(\mathcal{P})$ and $1 \leq i \leq art(p)$, we define a *position* $p[i]$. Given a set of predicates Σ, the set $\mathsf{ap}(\mathcal{P}, \Sigma)$ of *affected positions* (see (Calì, Gottlob, and Kifer 2013)) in $\mathcal{P}$ w.r.t. Σ is inductively defined as follows: (i) $p[i] \in \mathsf{ap}(\mathcal{P}, \Sigma)$, for $p \in \Sigma$ and $1 \leq i \leq art(p)$, and (ii) if there exists a rule $r \in \mathcal{P}$ s.t. a variable x appears in $body^{+}(r)$ only in affected positions and x appears in $head(r)$ in position π, then $\pi \in \mathsf{ap}(\mathcal{P}, \Sigma)$.

We call a predicate symbol p occurring in an RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ *bounded* in Π if $p \in \Sigma_{\mathsf{out}} \cup \Sigma_{\mathsf{re}} \cup \mathbf{B}_{\text{cn}}(\mathcal{T}, \Sigma_{\mathsf{out}})$. We now relax the previous safeness condition.

Definition 5. *An RLP $\Pi = (\mathcal{P}, \mathcal{T}, \Sigma_{\mathsf{out}}, \Sigma_{\mathsf{owa}}, \Sigma_{\mathsf{re}})$ fulfills* the relaxed safeness condition *if for each rule $r \in \mathcal{P}$: every variable in r occurs in some $p(\vec{u}) \in body^{+}(r)$, where p is a bounded predicate in Π and $q[i] \notin \mathsf{ap}(\mathcal{P}, \boldsymbol{B}_{cn}(\mathcal{T}, \Sigma_{\mathsf{out}}) \setminus \Sigma_{\mathsf{out}})$, for all $q \in \Sigma_{\mathsf{out}}$ and $1 \leq i \leq art(q)$.*

Algorithm 1, which was originally designed for checking answer set existence under the safeness condition in Definition 3, can be easily modified to cater to RLPs with DL theories under relaxed safeness. In particular, we need to modify the main procedure $hasAnswerSet$ to guess a candidate structure I over a set of constants whose number is at most exponential in the size of the input RLP. Recall that, in the original algorithm, this guess is limited to the constants that already appear in the input rules.

The previously-obtained complexity results hold also for RLPs with DL theories under relaxed safeness.

Theorem 6. *Table 2 provides correct complexity results for deciding the answer set existence for RLPs fulfilling the relaxed safeness condition.*

Discussion

In this paper, we have formalized and studied resilient logic programs as a novel hybrid language that addresses the shortcomings of the previous world-centric and entailment-centric approaches. There are several directions for future research. First, it is important to identify syntactic restrictions to lower the complexity of reasoning. The use of stratified negation for response predicates seems to be a promising approach, which we believe will allow us to avoid an NP oracle for checking the existence of responses in the computation of answer sets (roughly, eliminating the third-level NP oracle in the complexity results of this paper; see Theorem 2 and Table 2). The second direction is to study and implement various rewritings of DL-based RLPs into disjunctive Datalog, which has efficient reasoners (Leone et al. 2006). The presented translation of the fragment of RLPs with theories consisting of positive disjunctive rules provides the groundwork for this.

In this paper, DL ontologies are interpreted over Herbrand interpretations (whose domain is a fixed infinite set of constants). This was done for mathematical clarity of the semantics of RLPs, but it has some side-effects, e.g., ruling out ontology models with a finite domain. This can be easily fixed using a more complicated definition that handles two kinds of interpretations: Herbrand interpretations for rules, and ordinary first-order structures for DL ontologies.

Acknowledgements Funded by the Austrian Science Fund (FWF) projects P30360, P30873 and W1255.

References

Baader, F.; Horrocks, I.; Lutz, C.; and Sattler, U. 2017. *An Introduction to Description Logic*. Cambridge Univ. Press.

Bajraktari, L.; Ortiz, M.; and Šimkus, M. 2018. Combining rules and ontologies into Clopen knowledge bases. In *Proc. of AAAI 2018*.

Bárány, V.; Cate, B. T.; and Segoufin, L. 2015. Guarded negation. *J. ACM* 62(3):22:1–22:26.

Benedikt, M.; Bourhis, P.; ten Cate, B.; and Puppis, G. 2016. Querying visible and invisible information. In *Proc. of LICS 2016*, 297–306. ACM.

Bienvenu, M.; ten Cate, B.; Lutz, C.; and Wolter, F. 2014. Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. *ACM Trans. Database Syst.* 39(4):33:1–33:44.

Borgida, A. 1996. On the relative expressiveness of description logics and predicate logics. *Artif. Intell.* 82(1-2):353–367.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Calì, A.; Gottlob, G.; and Kifer, M. 2013. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res. (JAIR)* 48:115–174.

Eiter, T., and Polleres, A. 2003. Transforming co-np checks to answer set computation by meta-interpretation. In *Informal Proc. of Joint Conference on Declarative Programming (AGP 2003)*, 410–421.

Eiter, T.; Ianni, G.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2008. Combining answer set programming with description logics for the semantic web. *Artif. Intell.* 172(12-13):p. 1495.

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

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of ICLP/SLP 1988*, 1070–1080. MIT Press.

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.

Knorr, M.; Alferes, J. J.; and Hitzler, P. 2011. Local closed world reasoning with description logics under the well-founded semantics. *Artif. Intell.* 175(9-10):1528–1554.

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. Logic* 7(3):499–562.

Levy, A. Y., and Rousset, M. 1998. Combining horn rules and description logics in CARIN. *Artif. Intell.* 104(1-2):165–209.

Motik, B., and Rosati, R. 2010. Reconciling description logics and rules. *J. ACM* 57(5).

Motik, B.; Sattler, U.; and Studer, R. 2005. Query answering for OWL-DL with rules. *J. Web Sem.* 3(1):41–60.

Rosati, R. 2005. On the decidability and complexity of integrating ontologies and rules. *J. Web Sem.* 3(1):61–73.

Rosati, R. 2006. DL+log: Tight integration of description logics and disjunctive datalog. In *Proc. of KR 2006*. AAAI Press.

Rosati, R. 2007. The limits of querying ontologies. In *Proc. of ICDT 2007*, volume 4353 of *Lecture Notes in Computer Science*, 164–178. Springer.