Dealing with Inconsistency when Combining Ontologies and Rules using DL-Programs*

Jörg Pührer, Stijn Heymans, and Thomas Eiter

Institute of Information Systems 184/3
Vienna University of Technology
Favoritenstraße 9–11, A–1040 Vienna, Austria
{puehrer,heymans,eiter}@kr.tuwien.ac.at

Abstract. Description Logic Programs (DL-programs) have been introduced to combine ontological and rule-based reasoning in the context of the Semantic Web. A DL-program loosely combines a Description Logic (DL) ontology with a non-monotonic logic program (LP) such that dedicated atoms in the LP, called DL-atoms, allow for a bidirectional flow of knowledge between the two components. Unfortunately, the information sent from the LP-part to the DL-part might cause an inconsistency in the latter, leading to the trivial satisfaction of every query. As a consequence, in such a case, the answer sets that define the semantics of the DL-program may contain spoiled information influencing the overall deduction. For avoiding unintuitive answer sets, we introduce a refined semantics for DL-programs that is sensitive for inconsistency caused by the combination of DL and LP, and dynamically deactivates rules whenever such an inconsistency would arise. We analyze the complexity of the new semantics, discuss implementational issues and introduce a notion of stratification that guarantees uniqueness of answer sets.

1 Introduction

Recently, combinations of rule formalisms and ontologies (in particular *Description Logic (DL) theories* [1]) have gained increasing interest in the Semantic Web community. This is reflected in the Semantic Web Layer Architecture that envisions a Rules Layer complementing the Ontology Layer as a means for sophisticated representation and reasoning. A major issue in systems integrating rules and ontologies is how to realize the semantics of their combination.

A popular approach in this respect is *loose coupling*, i.e., the two components, ontology and rules, act separately but communicate via a well-defined interface. A realization of this approach is given by *Description Logic Programs* (*DL-programs*) [2], which combine a DL ontology with a logic program (LP). The semantics of the formalism is given by an extension of the *stable-model semantics* [3] that allows for using default negation for *non-monotonic reasoning*. DL-programs follow the answer set programming paradigm [4], where the program can be seen as a problem and the resulting stable models, called *answer*

* This work is partially supported by the Austrian Science Fund (FWF) projects P20840 and P21698, and by the EC FP7 project OntoRule (IST-2009-231875).

sets correspond to different solutions of this problem. The interface between the logic programming part and the Description Logic ontology, which is seen as a black box, is realized by dedicated atoms in the premises of the rules, called *DL-atoms*, which allow for a bidirectional exchange of information. The flow of information from the rules to the ontology provides a powerful tool, as results from the program can be used as assertions in the DL for further deduction.

However, it is possible that the assertions by which the ontology is extended cause an inconsistency. We say that in such a case the respective DL-atom is DL-inconsistent. As then the respective query is trivially true, we may end up with counterintuitive results, even though both the DL and the LP are perfectly consistent in separation.

In this work, we introduce a semantics, called the *DL-inconsistency tolerant semantics*, that aims at avoiding this effect by dynamically switching rules off whenever DL-inconsistency occurs. Thus, information derived from an inconsistency only cannot influence the reasoning in the LP-part.
The main contributions of this paper can be summarized as follows:

- We introduce a refined semantics for DL-programs that avoids unintuitive answer sets caused by DL-inconsistency and properly extends the answer set semantics for normal logic programs.
- We analyze the complexity of deciding whether a DL-program has an answer set under the new semantics. The problem turns out to be NExpTime-complete for many popular Description Logics. Moreover, we show that reasoning in our formalism can be reduced to reasoning in HEX-programs [5]. An implementation of our approach is targeted within the OntoRule project.
- We define a stratification property that guarantees the uniqueness of answer sets under the new semantics and ExpTime-completeness of deciding answer set existence. Based on these results, we present an algorithm for computing the answer set of a stratified program whenever one exists.

The remainder of the paper is organized as follows. In the next section, we give preliminaries on normal logic programs under the answer set semantics, Description Logics, and DL-programs. Section 3 explains the problem to be tackled in the paper, using an illustrative running example, and introduces the concept of DL-inconsistency. After introducing our new semantics in Section 4, we deal with various computational aspects in Section 5, discussing complexity and implementation issues, and introducing an adequate notion of stratification for the refined semantics. For brevity, we include here only selected proofs.

2 Preliminaries

2.1 Normal Programs under the Answer Set Semantics

An LP-signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ is a first-order signature such that $\mathcal{F}$ is a nonempty finite set of 0-ary function symbols (constants) and $\mathcal{P}$ is a nonempty finite set of predicate symbols. A *term* is any variable from a set of variables $\mathcal{V}$ or constant

symbol from $\mathcal{F}$. An *atom* is of form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ is a predicate symbol of arity $n \geq 0$ and $t_1, \ldots, t_n$ are terms. A *(classical) literal* l is an atom a or a negated atom $\neg a$. A *negation as failure literal* (or *NAF-literal*) is a literal l or a default-negated literal *not* l. A *normal rule* (simply, *rule*) r is of the form

$$a \leftarrow b_1, \ldots, b_k, not\ \ b_{k+1}, \ldots, not\ \ b_m\,, \qquad m \geq k \geq 0\,, \tag{1}$$

where $a, b_1, \ldots, b_m$ are classical literals. The literal a is the *head* of r, denoted by $H(r)$, and the conjunction $b_1, \ldots, b_k, not\ \ b_{k+1}, \ldots, not\ \ b_m$ is the *body* of r; its *positive* (resp., *negative*) part is $b_1, \ldots, b_k$ (resp., $not\ \ b_{k+1}, \ldots, not\ \ b_m$). We denote by $B(r)$ the set of body literals $B(r)^+ \cup B(r)^-$, where $B(r)^+ = \{b_1, \ldots, b_k\}$ and $B(r)^- = \{b_{k+1}, \ldots, b_m\}$. A *(normal) program* Π (over Σ) is a finite set of rules; Π is *positive* iff it is "*not* "-free.

The *Herbrand universe* of a program Π is the set $\mathrm{HU}(\Pi) \subseteq \mathcal{F}$ of all constant symbols in Π (if no such symbol exists, $\mathrm{HU}(\Pi) = \{c\}$ for an arbitrary constant symbol c from $\mathcal{F}$). Moreover, the *Herbrand base* of a program Π, denoted $\mathrm{HB}(\Pi)$, is the set of all ground (classical) literals with predicate symbols appearing in Π and constant symbols from $\mathrm{HU}(\Pi)$. The notions of *ground terms, atoms, literals etc.* are as usual. We denote by $gr_S(\Pi)$ the grounding of Π w.r.t a set $S \subseteq \mathcal{F}$ of constants, i.e., the ground rules originating from rules in Π by replacing, per rule, each variable by each possible combination of constants in S.

A set of literals $X \subseteq \mathrm{HB}(\Pi)$ is *consistent* iff $\{p, \neg p\} \not\subseteq X$ for every atom $p \in \mathrm{HB}(\Pi)$. An *interpretation* I relative to Π is a consistent subset of $\mathrm{HB}(\Pi)$. I *satisfies* the positive (resp., negative) body of a rule r, symbolically $I \models B(r)^+$ (resp., $I \models B(r)^-$), if $B(r)^+ \subseteq I$ (resp., $I \cap B(r)^- = \emptyset$). I *satisfies* the body of r, denoted $I \models B(r)$, if $I \models B(r)^+$ and $I \models B(r)^-$. I *satisfies* a rule r, symbolically $I \models r$, if $H(r) \in I$ whenever $I \models B(r)$. An interpretation $I \subseteq \mathrm{HB}(\Pi)$ is a *model* of a program Π, denoted by $I \models \Pi$, if every $r \in gr_{\mathrm{HU}(\Pi)}(\Pi)$ is satisfied by I. An *answer set* of a positive program Π is the least model of Π w.r.t. $\subseteq$.

Answer sets were traditionally defined in terms of the *Gelfond-Lifschitz reduct* [3]. We here use the equivalent definition of answer sets by means of the *FLP-reduct* of Π relative to an interpretation $I \subseteq \mathrm{HB}(\Pi)$, denoted Π^I_{FLP}, following Faber, Leone, and Pfeifer [6]. It has been introduced for an intuitive handling of aggregates in answer set programming and is a special case of the *t-reduct* that we use for defining the semantics introduced in this paper. The FLP-reduct of Π relative to an interpretation $I \subseteq \mathrm{HB}(\Pi)$, denoted Π^I_{FLP}, is the set of rules $r \in gr_{\mathrm{HU}(\Pi)}(\Pi)$ such that $I \models B(r)$. Then, I is an answer set of Π iff I is a minimal model of Π^I_{FLP}.

Example 1. As a simple example, consider the program P that consists of the three rules $c(t)$, $a(t) \leftarrow not\ b(t)$, and $b(t) \leftarrow c(t), not\ a(t)$. It has two answer sets, viz. $I_1 = \{c(t), a(t)\}$ and $I_2 = \{c(t), b(t)\}$.

2.2 Description Logics

The approach to resolving inconsistencies caused by DL-atoms is to a large extent independent of a specific Description Logic (DL) [1]. For a particular DL knowledge base Φ, we will assume that

– it is defined over a signature $\Sigma_o = \langle \mathcal{F}, \mathcal{P}_o \rangle$ with individuals from $\mathcal{F}$ and concept and role names from $\mathcal{P}_o$,
– it is able to deal with ground unary or binary literals, i.e., an expression $\Phi \cup \{C(a), \neg C(a), R(a,b), \neg R(a,b)\}$ is well-defined for unary (binary) predicates C (R) from $\mathcal{P}_o$ and individuals a from $\mathcal{F}$, and
– it defines an entailment relation $\models$ such that $\Phi \models Q(\boldsymbol{t})$ is defined for *DL-queries* $Q(\boldsymbol{t})$ and indicates that all models of Φ satisfy $Q(\boldsymbol{t})$.

A *DL-query* $Q(\mathbf{t})$ is either

(a) a concept inclusion axiom F or its negation $\neg F$; or
(b) of the forms $C(t)$ or $\neg C(t)$, where C is a concept and t is a term; or
(c) of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role and t_1, t_2 are terms.

Exemplary DLs that satisfy these minimum requirements are $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SROIQ}(\mathbf{D})$ which provide the logical underpinnings of the Web ontology languages OWL DL and OWL 2 (see [7; 8; 9] for further background). In what follows we assume that the reader is familiar with standard DL syntax.

Example 2 (Product Database). As our running example, we will adapt an example that has been used previously in the context of DL-programs [10].

A small computer store obtains its hardware from several vendors. It uses the following DL knowledge base Φ_{ex}, which contains information about the product range that is provided by each vendor. For some parts, a shop may already be contracted as supplier and shops which are known to be disapproved for some reason can never become an actual supplier.

$$\geq 1\ supplier \sqsubseteq Shop; \quad \top \sqsubseteq \forall supplier.Part;$$
$$\exists supplier.\top \sqcap disapproved \sqsubseteq \bot;$$
$$Shop(s_1); \quad Shop(s_2); \quad Shop(s_3); \quad disapproved(s_2);$$
$$Part(harddisk); \quad Part(cpu); \quad Part(case);$$
$$provides(s_1, cpu); \quad provides(s_1, case); \quad provides(s_2, cpu);$$
$$provides(s_3, harddisk); \quad provides(s_3, case);$$
$$supplier(s_3, case);$$

Here, the first two axioms determine *Shop* and *Part* as domain and range of the property *supplier*, respectively, while the third axiom constitutes the incongruity between shops that are contracted as supplier but are explicitly disapproved.

2.3 DL-Programs

Syntax A signature $\Sigma = \langle \mathcal{F}, \mathcal{P}_o, \mathcal{P}_p \rangle$ for DL-programs consists of a set $\mathcal{F}$ of 0-ary function symbols and sets $\mathcal{P}_o$, $\mathcal{P}_p$ of predicate symbols such that $\Sigma_o = \langle \mathcal{F}, \mathcal{P}_o \rangle$ is a DL-signature and $\Sigma_p = \langle \mathcal{F}, \mathcal{P}_p \rangle$ is an LP-signature.

Informally, a DL-program consists of a Description Logic ontology Φ over Σ_o and a normal program Π over Σ_p, which may contain queries to Φ. Roughly, in such a query, it is asked whether a certain Description Logic formula or its negation logically follows from Φ or not.

A *DL-atom* $a(\mathbf{t})$ has the form

$$\mathrm{DL}[S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m; Q](\mathbf{t}), \qquad m \geq 0, \tag{2}$$

where each S_i is either a concept or a role predicate from $\mathcal{P}_o$, $op_i \in \{\uplus, \cup, \cap\}$, p_i is a unary, resp. binary, predicate symbol from $\mathcal{P}_p$, and $Q(\mathbf{t})$ is a DL-query. We call $\gamma = S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m$ the *input signature* and $p_1, \ldots, p_m$ the *input predicate symbols* of $a(\mathbf{t})$. Moreover, literals over input predicate symbols are *input literals*. Intuitively, $op_i = \uplus$ (resp., $op_i = \cup$) increases S_i (resp., $\neg S_i$) by the extension of p_i, while $op_i = \cap$ constrains S_i to p_i. A *DL-rule* r has the form (1), where any literal $b_1, \ldots, b_m \in B(r)$ may be a DL-atom. A *DL-program* $\mathcal{KB} = (\Phi, \Pi)$ consists of a DL ontology Φ and a finite set of DL-rules Π.

Example 3. Consider the DL-program $\mathcal{KB}_{ex} = (\Phi_{ex}, \Pi_{ex})$, with Φ_{ex} as in Example 2 and Π_{ex} given as follows, choosing not-deterministically a vendor for each needed part:

(1) $needed(cpu)$; $needed(harddisk)$; $needed(case)$;
(2) $alreadyContracted(P) \leftarrow \mathrm{DL}[; supplier](S, P), needed(P)$;
(3) $offer(S, P) \leftarrow \mathrm{DL}[; provides](S, P), needed(P), not\ alreadyContracted(P)$;
(4) $chosen(S, P) \leftarrow offer(S, P), not\ notChosen(S, P)$;
(5) $notChosen(S, P) \leftarrow offer(S, P), not\ chosen(S, P)$;
(6) $supplied(S, P) \leftarrow \mathrm{DL}[supplier \uplus chosen; supplier](S, P), needed(P)$;
(7) $anySupplied(P) \leftarrow supplied(S, P), needed(P)$;
(8) $fail \leftarrow not\ fail, needed(P), not\ anySupplied(P)$.

Rule (2) extracts information on which parts already have a fixed vendor assigned from the DL, whereas Rule (3) imports the available offers for the needed parts not yet assigned. Rules (4)-(5) nondeterministically decide whether an offer should be chosen. Rule (6) summarizes the purchasing results by first sending the chosen assignments of vendors and parts from the LP-part to the ontology, and then querying for the overall *supplier* relation. Finally, Rules (7)-(8) ensure that for every needed part there is a vendor chosen who supplies it. Note that Rule (8) acts as a constraint where the occurrences of the auxiliary atom *fail* in both, head and positive body, prevents all interpretations containing $needed(t)$ but not $anySupplied(t)$ for any term t from being an answer set. As we will see in Section 3, Φ_{ex} has one intended and one counterintuitive answer set.

Semantics In the sequel, let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program over $\Sigma = \langle \mathcal{F}, \mathcal{P}_o, \mathcal{P}_p \rangle$. By $gr(\Pi)$ we denote the grounding of Π w.r.t $\mathcal{F}$, i.e., the set of ground rules originating from DL-rules in Π by replacing, per DL-rule, each variable by each possible combination of constants in $\mathcal{F}$.

An *interpretation* I (over Σ_p) is a consistent subset of literals over Σ_p. We say that I satisfies a classical literal l under Φ, denoted $I \models^{\Phi} l$, iff $l \in I$, and a ground DL-atom $a = DL[S_1 op_1\ p_1, \ldots, S_m op_m p_m; Q](\mathbf{c})$ under Φ, denoted $I \models^{\Phi} a$, if $\Phi \cup \tau^I(a) \models Q(\mathbf{c})$, where the extension $\tau^I(a)$ of a under I is defined as $\tau^I(a) = \bigcup_{i=1}^{m} A_i(I)$ such that

- $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \cup$;
- $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}$, for $op_i = \cap$.

We say that I *satisfies* the positive (resp., negative) body of a ground DL-rule r under Φ, symbolically $I \models^\Phi B(r)^+$ (resp., $I \models^\Phi B(r)^-$), if $I \models^\Phi l$ (resp., $I \not\models^\Phi l$) for all $l \in B(r)^+$ (resp., $l \in B(r)^-$). I *satisfies* the body of r under Φ, denoted $I \models^\Phi B(r)$, whenever $I \models^\Phi B(r)^+$ and $I \models^\Phi B(r)^-$. I satisfies a ground DL-rule r under Φ, symbolically $I \models^\Phi r$, if $I \models^\Phi H(r)$ whenever $I \models^\Phi B(r)$. I is a model of a DL-program $\mathcal{KB} = (\Phi, \Pi)$, denoted $I \models \mathcal{KB}$, iff $I \models^\Phi r$ for all $r \in gr(\Pi)$. We say $\mathcal{KB}$ is *satisfiable* (resp., *unsatisfiable*) iff it has some (resp., no) model.

In this paper, we base the answer set semantics of DL-programs on the Faber-Leone-Pfeifer reduct, rather than on the Gelfond-Lifschitz reduct.

Definition 1. *Let $\Sigma = \langle \mathcal{F}, \mathcal{P}_o, \mathcal{P}_p \rangle$ be a signature for DL-programs, Φ a DL knowledge base over $\langle \mathcal{F}, \mathcal{P}_o \rangle$, Π a set of ground DL-rules over $\Sigma_p = \langle \mathcal{F}, \mathcal{P}_p \rangle$, and I an interpretation over Σ_p. The* FLP-reduct $\Pi^{I,\Phi}_{FLP}$ *of Π under Φ relative to I is the set of rules $r \in \Pi$ such that $I \models^\Phi B(r)$. Moreover, the* FLP-reduct $\mathcal{KB}^I_{FLP}$ *of a (possibly non-ground) DL-program $\mathcal{KB} = (\Phi, \Pi)$ relative to I is given by $gr(\Pi)^{I,\Phi}_{FLP}$.*

Definition 2. *Let $\mathcal{KB}$ be a DL-program over $\Sigma = \langle \mathcal{F}, \mathcal{P}_o, \mathcal{P}_p \rangle$. An interpretation I over Σ_p is an* answer set *of $\mathcal{KB}$ if it is a minimal model of $\mathcal{KB}^I_{FLP}$. The set of all answer sets of $\mathcal{KB}$ is denoted by* AS($\mathcal{KB}$).

We use this answer set semantics (we will sometimes refer to it as *FLP-semantics*), as it naturally handles DL-atoms which are not *monotonic*.

Definition 3. *For a DL-program $\mathcal{KB} = (\Phi, \Pi)$, a ground DL-atom l is* monotonic *relative to $\mathcal{KB}$, if for all interpretations I, J with $I \subseteq J$, $I \models^\Phi l$ implies $J \models^\Phi l$. $\mathcal{KB}$ is* monotonic *if $gr(\Pi)$ contains only DL-atoms that are monotonic relative to $\mathcal{KB}$.*

It was shown in [5] that for DL-programs that do not employ the $\cap$ operator, the FLP-semantics coincides with strong answer set semantics, as originally introduced for DL-programs [2] using the Gelfond-Lifschitz reduct. Note that this operator is rarely used in practice and can in many cases (e.g., for *t-stratified* DL-programs, cf. Section 5) be removed by simple translations.

We will later refer to the class of *positive DL-programs*, defined in [2] as follows.

Definition 4. *A DL-program $\mathcal{KB}$ is* positive*, if it is monotonic and $B(r)^- = \emptyset$ for each rule $r \in \Pi$.*

Note that a DL-atom a that does not employ the operator $\cap$ is always monotonic as $I \subseteq J$ implies $\tau^I(a) \subseteq \tau^J(a)$.

3 Inconsistency when Combining Ontologies and Rules

We will now look at the semantics of our example DL-program in order to illustrate the core problem we want to tackle in our approach.

Example 4. $\mathcal{KB}_{ex}$ has two answer sets, I_1 and I_2, both containing the same atoms of predicates *needed*, *offer*, *alreadyContracted*, and *anySupplied*:

$$\begin{aligned} I' = \{\ & needed(cpu), needed(harddisk), needed(case), alreadyContracted(case), \\ & offer(s_1, cpu), offer(s_2, cpu), offer(s_3, harddisk), \\ & anySupplied(cpu), anySupplied(harddisk), anySupplied(case)\} \end{aligned}$$

The remaining atoms of I_1 are given by

$$\begin{aligned} I_1 \setminus I' = \{ & chosen(s_1, cpu), chosen(s_3, harddisk), notChosen(s_2, cpu), \\ & supplied(s_1, cpu), supplied(s_3, harddisk), supplied(s_3, case)\}\ , \end{aligned}$$

expressing a solution where the cpu is provided by shop s_1, whereas harddisk and case are delivered by vendor s_3.

The second answer set might seem surprising at first sight:

$$\begin{aligned} I_2 \setminus I' = \{ & chosen(s_2, cpu), chosen(s_3, harddisk), notChosen(s_2, cpu), \\ & supplied(s_1, cpu), supplied(s_1, harddisk), supplied(s_1, case), \\ & supplied(s_2, cpu), supplied(s_2, harddisk), supplied(s_2, case), \\ & supplied(s_3, cpu), supplied(s_3, harddisk), supplied(s_3, case), \\ & supplied(cpu, cpu), supplied(cpu, harddisk), supplied(cpu, case), \\ & supplied(harddisk, cpu), supplied(harddisk, harddisk), \\ & supplied(harddisk, case), supplied(case, cpu), supplied(case, harddisk), \\ & supplied(case, case)\} \end{aligned}$$

Apparently a situation is described in which each of the shops s_1, s_2, and s_3 supplies each of the needed hardware parts *cpu*, *case*, and *harddisk*, although the intention was that only a single shop supplies one part. Moreover, we also have atoms like $supplied(cpu, harddisk)$ in I_2, completely lacking intuition, as the first argument of predicate *supplied* is supposed to refer to vendors only. The reason for the unintuitive results lies in an inconsistency emerging in the combination of the ontology and the logic programming part of $\mathcal{KB}_{ex}$. Note that atom $chosen(s_2, cpu) \in I_2$ suggests that shop s_2 has been chosen to deliver the cpu, although this shop is identified as disapproved in the DL-part (cf. Example 2). Consider any ground instance a' of DL-atom $a = \mathrm{DL}[supplier \uplus chosen; supplier](S, P)$ in Rule (6) of extended logic program Π_{ex}. We then have $\tau^I(a') = \{supplier(\mathbf{e}) \mid chosen(\mathbf{e}) \in I\}$ and therefore $supplier(s_2, cpu) \in \tau^I(a')$. As a consequence, $\Phi \cup \tau^I(a')$ is inconsistent since $\neg supplier(s_2, cpu)$ follows from the axioms $\exists supplier.\top \sqcap disapproved \sqsubseteq \bot$ and $disapproved(s_2)$ in Φ_{ex}. Due to this inconsistency every ground instance of a is true under I_2.

Whenever information, passed from the logic programming part Π to the ontology Φ of a DL-Program, is inconsistent with Φ, unintuitive answer sets may

arise as a consequence of trivial satisfaction of DL-atoms. In such cases we call the respective DL-atom *DL-inconsistent*.

Definition 5. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program and I an interpretation relative to Π. A ground DL-atom $a = \mathrm{DL}[\gamma; Q](\mathbf{c})$ is* DL-consistent *under I w.r.t. Φ, if (1) $\Phi \models Q(\mathbf{c})$ or (2) $\Phi \cup \tau^I(a)$ is consistent, otherwise a is* DL-inconsistent *under I w.r.t. Φ.*

Intuitively, we are interested in avoiding using rules that have DL-inconsistent atoms in their bodies. Note that we use a notion of "inconsistence" that pertains to updates of the ontology: if some atom $Q(\mathbf{c})$ is entailed by the original ontology, we assume it is DL-consistent, even if updates via γ make the ontology inconsistent. Indeed, if $\Phi \models Q(\mathbf{c})$, we also have $\Phi \cup \tau^I(a) \models Q(\mathbf{c})$ for any update $\tau^I(a)$ due to monotonicity of usual Description Logics. If we would not take this case into account, we would disregard the whole rule (as seen in Definition 6).

4 DL-Inconsistency Tolerant Semantics

In what follows we introduce and discuss a refined semantics for DL-programs that limits the negative side effects of DL-inconsistency. The central idea is to deactivate a rule whenever a DL-atom contained in its body becomes DL-inconsistent, in order to behave *tolerant* in the sense that flawed information does not influence the derived results. This way literals with unexpected argument types such as $\mathit{supplied}(\mathit{cpu}, \mathit{harddisk})$ in Example 4, can be avoided in the information flow from the ontology to the logic program.

Definition 6. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program and I an interpretation. I* t-satisfies *the body of a ground DL-rule r under Φ, denoted $I \models_t^\Phi B(r)$ if $I \models^\Phi B(r)$ and all DL-atoms in $B(r)$ are DL-consistent under I w.r.t. Φ. Moreover, I* t-satisfies *r under Φ, symbolically $I \models_t^\Phi r$, if $I \models_t^\Phi B(r)$ implies that $I \models^\Phi H(r)$. I is a* t-model *of a set Q of ground DL-rules under Φ denoted $I \models_t^\Phi Q$ if $I \models_t^\Phi r$ for all $r \in Q$. Finally, I is a* t-model *of $\mathcal{KB}$, denoted $I \models_t \mathcal{KB}$, if $I \models_t^\Phi gr(\Pi)$.*

Note that every model of $\mathcal{KB}$ is also a t-model of $\mathcal{KB}$. Moreover, if DL-atoms occur only in the negative bodies of rules in Π, also the converse holds. The reason for the latter is that a rule that is not applicable under DL-inconsistency tolerant semantics only because of a DL-inconsistent DL-atom $a \in B(r)^-$ for some rule $r \in \Pi$ would also not be applicable under standard semantics as a would be satisfied as a consequence of DL-inconsistency.

Example 5. Consider the ground instantiation

$$r = \mathit{supplied}(\mathit{cpu}, \mathit{harddisk}) \leftarrow \mathrm{DL}[\mathit{supplier} \uplus \mathit{chosen};\ \mathit{supplier}](\mathit{cpu}, \mathit{harddisk}),\ \mathit{needed}(\mathit{harddisk})$$

of Rule (6) of our running example. For interpretation I_2, as defined in Example 4, we have that $I_2 \models^\Phi B(r)$ but, as the DL-atom in $B(r)$ is DL-inconsistent under I_2 w.r.t. Φ_{ex}, it holds that $I_2 \not\models_t^\Phi B(r)$. As $H(r) \in I_2$, both $I_2 \models^\Phi r$ and $I_2 \models_t^\Phi r$. More general, since I_2 is a model of $\mathcal{KB}_{ex}$ it is also a t-model of $\mathcal{KB}_{ex}$. However, as we will see next, I_2 is not a *t-answer set* of $\mathcal{KB}_{ex}$.

For defining the notion of a t-answer set, we first give a modified version of the FLP-reduct, called *t-reduct*.

Definition 7. *Let $\Sigma = \langle \mathcal{F}, \mathcal{P}_o, \mathcal{P}_p \rangle$ be a signature for DL-programs, Φ a DL knowledge base over $\langle \mathcal{F}, \mathcal{P}_o \rangle$, Π a set of ground DL-rules over $\Sigma_p = \langle \mathcal{F}, \mathcal{P}_p \rangle$, and I an interpretation over Σ_p. The* t-reduct $\Pi_t^{I,\Phi}$ *of Π under Φ relative to I is the set of rules $r \in \Pi$ such that $I \models_t^{\Phi} B(r)$. Moreover, the* t-reduct $\mathcal{KB}_t^I$ *of a (possibly non-ground) DL-program $\mathcal{KB} = (\Phi, \Pi)$ relative to I is given by $gr(\Pi)_t^{I,\Phi}$.*

Definition 8. *Let $\mathcal{KB}$ be a DL-program. An interpretation I is a* t-answer set *of $\mathcal{KB}$, if I is a subset-minimal t-model of $\mathcal{KB}_t^I$. The set of all t-answer sets of $\mathcal{KB}$ is denoted by* $\mathrm{AS}^{\mathrm{t}}(\mathcal{KB})$.

Example 6. For the program $\mathcal{KB}_{ex}$ of the product database example, the only t-answer set is given by interpretation I_1, as defined in Example 4. As stated in Example 5, I_2 is a t-model of $\mathcal{KB}_{ex}$; however, I_2 is not a minimal t-model of $(\mathcal{KB}_{ex})_t^{I_2}$, as required in Definition 8 for being a t-answer set. In fact, the ground instance of Rule (6) in Example 5 is not contained in $(\mathcal{KB}_{ex})_t^{I_2}$. Therefore, we can remove the head of the rule, atom $supplied(cpu, harddisk)$, from I_2 such that the resulting interpretation I_2' is still a t-model of $\mathcal{KB}_{ex}$.

Whenever no DL-atoms are present in a DL-program $\mathcal{KB} = (\Phi, \Pi)$, DL-inconsistency tolerant semantics reduces to answer set semantics of the ordinary logic program Π. Therefore, the next result is a proper extension to a similar one that is folklore for standard logic programs.

Theorem 1. *For every t-answer set I of a DL-program $\mathcal{KB}$, I is a minimal t-model of $\mathcal{KB}$.*

Note that the converse does not generally hold. E.g., consider the set

$$I_3 = I' \cup \{chosen(s_2, cpu), chosen(s_3, harddisk), notChosen(s_2, cpu), \\ notChosen(s_2, harddisk)\} ,$$

where I' is given as in Example 4. I_3 is a minimal t-model of $\mathcal{KB}_{ex}$ but, since the ground instantiation

$$fail \leftarrow not\ fail, needed(cpu), not\ anySupplied(cpu)$$

of Rule (8) from our example is not contained in $(\mathcal{KB}_{ex})_t^{I_3}$, we can remove atom $anySupplied(cpu)$ from I_3 such that the resulting interpretation I_3' is still a t-model of $(\mathcal{KB}_{ex})_t^{I_3}$. Consequently, by Definition 8, I_3 is no t-answer set of $\mathcal{KB}_{ex}$.

The next result relates the refined semantics to the FLP-semantics.

Proposition 1. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a monotonic DL-program and let I be an answer set of $\mathcal{KB}$. If all DL-atoms in $gr(\Pi)$ are DL-consistent under I w.r.t. Φ, then I is a t-answer set of $\mathcal{KB}$.*

Note that DL-programs with no occurrences of the $\Cap$ operator are monotonic and, as remarked in Section 2, this operator can typically be avoided in applications.

While counterintuitive literals a là *supplied*(*cpu*, *harddisk*) cannot occur in a t-answer set, Proposition 1 suggests that results that are intuitive are preserved under the refined semantics, as answer sets of a DL-program where inconsistency is immaterial are selected. On the other hand, a DL-program may have t-answer sets that do not correspond to any answer set (due to inconsistency avoidance).

Example 7. Consider the DL-program $\mathcal{KB} = (\Phi, \Pi)$ where $\Phi = \{\neg C(a)\}$ and $\Pi = \{\, p(a);\ \ \mathit{fail} \leftarrow \mathit{not}\ \mathit{fail}, \mathrm{DL}[C \uplus p;\, C](a)\,\}$. Clearly, $\mathcal{KB}$ has no answer set, as the DL-atom in Π is DL-inconsistent; its single t-answer set is $I = \{p(a)\}$.

5 Computational Aspects

Translation to FLP Semantics The DL-inconsistency tolerant semantics of DL-programs can be simulated by the FLP semantics as in Definition 2 using a linear rule-by-rule transformation $\rho(\cdot)$ on generalized normal programs, defined as

$$\begin{aligned}
\rho(\Pi) =& \{\rho(r) \mid r \in \Pi\} \cup \\
& \{a' \leftarrow \mathrm{DL}[\gamma;\, \top \sqsubseteq \bot], \mathit{not}\ \mathrm{DL}[;\, Q](\mathbf{t}) \mid r \in \Pi, \mathit{not}\ a \in A(r)\}, \quad \text{where} \\
\rho(r) =& H(r) \leftarrow B(r) \cup A(r), \quad \text{and} \\
A(r) =& \{\mathit{not}\ a' \mid a = \mathrm{DL}[\gamma;\, Q](\mathbf{t}) \in B(r)\}.
\end{aligned}$$

In the translation for each DL-atom $a = \mathrm{DL}[\gamma;\, Q](\mathbf{t})$ occurring in the body of a rule r, we add a new atom a' to the negative body of r and a rule that deduces a' exactly when $I \models^{\Phi} \mathrm{DL}[\gamma;\, \top \sqsubseteq \bot]$ and $I \not\models^{\Phi} \mathrm{DL}[;\, Q](\mathbf{t})$ for some interpretation I, i.e., when $\Phi \cup \tau^I(a)$ is inconsistent and $\Phi \not\models Q(\mathbf{c})$, and thus a is DL-consistent. Deduction of a' thus causes the body of the transformed rule to be false under FLP-semantics corresponding exactly the case where the atom a is DL-inconsistent. Thus for a rule r in Π under a DL-inconsistency tolerant semantics and its corresponding rule $\rho(r)$ in the transformed program $\rho(\Pi)$ under FLP-semantics, we have that r and $\rho(r)$ have bodies whose truth values correspond under the respective semantics, thus effectively mimicking the DL-inconsistency tolerant semantics with the FLP-semantics.

Theorem 2. *For every DL-program* $\mathcal{KB} = (\Phi, \Pi)$, $\mathrm{AS}^{\mathrm{t}}(\mathcal{KB}) = \{I \cap \mathrm{HB}(\Pi) \mid I \in \mathrm{AS}((\Phi, \rho(\Pi)))\}$.

By means of this translation, the t-answer sets of $\mathcal{KB}$ can be computed utilizing DLVHEX, a solver for *non-monotonic logic programs* admitting *higher-order atoms* and *external atoms*, or *HEX-programs* for short [5], that have a semantics based on the FLP-reduct. A plug-in for evaluating DL-programs, without the $\Cap$ operator, is available for DLVHEX that gives access to the DL-knowledge base by means of a third-party DL-reasoner [11; 12].

Due to the close relationship to HEX-programs, results on their computational complexity carry over to DL-inconsistency tolerant semantics of DL-programs. In particular, as corollaries of Theorem 7 and 8 in [5], due to the existence of transformation $\rho(\cdot)$, we obtain the following two results.

Theorem 3. *Given a DL-program $\mathcal{KB} = (\Phi, \Pi)$, where query answering in Φ is in complexity class* C*, deciding whether $\mathcal{KB}$ has a t-answer set is in* $\text{NEXPTIME}^{\text{C}}$.

Theorem 4. *Given a DL-program $\mathcal{KB} = (\Phi, \Pi)$, where query answering in Φ is in* ExpTime*, deciding whether $\mathcal{KB}$ has a t-answer set is* NExpTime*-complete.*

Hardness in Theorem 4 follows from the special case of DL-programs without any DL-atoms, for which the DL-inconsistency tolerant semantics reduces to the standard answer set semantics of normal logic programs. It is known that answer set existence for this class of programs is NExpTime-complete. On the other side, membership follows again from the translation to HEX-programs, as it is known that checking the answer sets of HEX-programs is NExpTime-complete under the restriction that the external atoms can be evaluated in exponential time [5].

The result is especially interesting as query answering is in ExpTime for many important DLs such as the basic DL $\mathcal{ALC}$, the DL underlying OWL-Lite ($\mathcal{SHIF}$), and the DLs corresponding to the fragments OWL 2 EL, OWL 2 RL, OWL 2 QL of the upcoming standard for a Web Ontology Language [9].

Another important aspect of the complexity results is that for DL-programs, reasoning under DL-inconsistency tolerant semantics is not harder than under the FLP-semantics.

Stratification Eiter et al. [2] defined an iterative least model semantics for DL-programs that have a certain stratification property (which we will here refer to as *standard stratification*). The idea of stratification is to layer a program into a number of ordered strata that can be efficiently evaluated one-by-one where lower strata do not depend on higher strata.

A DL-program $\mathcal{KB}$ which is standard stratified has at most one answer set that coincides with its iterative least model and conversely, if $\mathcal{KB}$ has an iterative least model it coincides with the unique answer set of $\mathcal{KB}$. However, a DL-program that is standard stratified may have multiple t-answer sets. Too see this, note that a positive DL-program always has a standard stratification with a single stratum. Consider, e.g., the DL-program $\mathcal{KB} = (\Phi, \Pi)$, with

$$\begin{aligned}\Pi = \{\ & h(c), \\ & a(c) \leftarrow \text{DL}[B \uplus b, H \ \uplus\ h;\ H](c), \\ & b(c) \leftarrow \text{DL}[A \uplus a, H \ \uplus\ h;\ H](c)\}\ ,\end{aligned}$$

where $\Phi \models A(c)$ and $\Phi \models B(c)$. This program has two t-answer sets, viz. $I_1 = \{a(c), h(c)\}$ and $I_2 = \{b(c), h(c)\}$.

In the following, we define a different kind of stratification (which we call *t-stratification*) that guarantees a unique t-answer set iff the respective t-stratified program has a t-answer set. The major difference to standard stratification is to enforce that the information necessary for evaluating DL-atoms must be already available on a strictly lower stratum then the current one during a computation.

Definition 9. *A* t-stratification *of a DL-program $\mathcal{KB} = (\Phi, \Pi)$ is a mapping $\mu : \text{HB}(\Pi) \cup \text{D}(\Pi) \to \{0, 1, \ldots, k\}$, where* $\text{D}(\Pi)$ *is the set of DL-atoms occurring in $gr(\Pi)$, such that*

(i) *for each* $r \in gr(\Pi)$, $\mu(H(r)) \geq \mu(l')$ *for all* $l' \in B(r)^+$, $\mu(H(r)) > \mu(l')$ *for all* $l' \in B(r)^-$, *and* $\mu(H(r)) > \mu(l')$ *for each DL-atom* $l' \in B(r)$, *and*
(ii) $\mu(a) \geq \mu(l)$ *for each input literal* l *of each DL-atom* $a \in \mathrm{D}(\Pi)$.

We call $k \geq 0$ the length of μ. For every $i \in \{0, \ldots, k\}$, we then define the DL-programs $\mathcal{KB}_{\mu,i}$ as (Φ, Π_i), where $\Pi_i = \{r \in gr(\Pi) \mid \mu(H(r)) = i\}$ and $\mathcal{KB}^*_{\mu,i}$ as $(\Phi, \Pi^*_{\mu,i})$ where $\Pi^*_{\mu,i} = \{r \in gr(\Pi) \mid \mu(H(r)) \leq i\}$. Likewise, we define $\mathrm{HB}_{\mu,i}(\Pi)$ (resp., $\mathrm{HB}^*_{\mu,i}(\Pi)$) as the set of all $l \in \mathrm{HB}(\Pi)$ such that $\mu(l) = i$ (resp., $\mu(l) \leq i$). We say that a DL-program $\mathcal{KB}$ is *t-stratified*, if it has a t-stratification μ of length $k \geq 0$. It is easy to see that for DL-programs without DL-atoms, t-stratification reduces to standard stratification of logic programs. Moreover, checking whether a DL-program is t-stratified and computing a t-stratification can be done by modified algorithms for standard stratification in linear time.

Note that by Definition 9, $\mathcal{KB}^*_{\mu,0}$ is always a positive DL-program without DL-atoms. Consequently, Π_0 coincides with a positive logic program, for which DL-inconsistency tolerant semantics coincides with the answer set semantics of logic programs. Therefore, the following proposition holds.

Proposition 2. *Let* $\mathcal{KB}$ *be a DL-program* $\mathcal{KB} = (\Phi, \Pi)$ *with t-stratification* μ. *Then,* $\mathcal{KB}^*_{\mu,0}$ *has a unique minimal t-model that is also the unique t-answer set of* $\mathcal{KB}^*_{\mu,0}$.

Next we want to establish uniqueness of t-answer sets for arbitrary strata.

Lemma 1. *Let* $\mathcal{KB}$ *be a DL-program* $\mathcal{KB} = (\Phi, \Pi)$ *with t-stratification* μ. *If* I_1 *and* I_2 *are t-answer sets of* $\mathcal{KB}^*_{\mu,i}$ *for* $i \geq 0$, *then* $I_1 = I_2$.

As a consequence of this lemma and Proposition 2, we get the next result.

Theorem 5. *Let* $\mathcal{KB}$ *be a t-stratified DL-program* $\mathcal{KB} = (\Phi, \Pi)$. *If* $\mathcal{KB}$ *has a t-answer set, then this t-answer set is unique.*

As can be seen in the next result, the t-answer set of a t-stratified DL-program is *compositional* in the sense that, roughly speaking, we get t-answer sets for the part of the DL-program that is below a certain stratum, if we remove all atoms of higher strata from I.

Theorem 6. *Let* $\mathcal{KB}$ *be a DL-program* $\mathcal{KB} = (\Phi, \Pi)$ *with t-stratification* μ. *If* I *is a t-answer set of* $\mathcal{KB}^*_{\mu,i}$ *for* $i > 0$, *then* $I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)$ *is a t-answer set of* $\mathcal{KB}^*_{\mu,i-1}$.

Approaching from this result, we aim at computing the t-answer set I of $\mathcal{KB}$ step-by-step, starting with $I \cap \mathrm{HB}^*_{\mu,0}(\Pi)$ and extending the interpretation one stratum a time until we reach $I = I \cap \mathrm{HB}^*_{\mu,k}(\Pi)$. Hence, we define a series of sets $\Delta_{i,h}$ for each stratum i, that can be seen as the results of repeatedly applying a consequence operator.

Definition 10. *Let* $\mathcal{KB}$ *be a DL-program* $\mathcal{KB} = (\Phi, \Pi)$ *with t-stratification* μ *and* I_{i-1} *a t-answer set of* $\mathcal{KB}^*_{\mu,i-1}$ *for some* $i > 0$. *We define sets of literals* $\Delta_{i,h}$ *for* $h \geq 0$ *as follows:*

(i) $\Delta_{i,0} = \emptyset$ *and*
(ii) $\Delta_{i,m} = \bigcup_{o<m} \Delta_{i,o} \cup \{H(r) \mid \mu(H(r)) = i, I_{i-1} \cup \Delta_{i,m-1} \models_t^{\Phi} B(r)\}$ *for* $m > 0$.

As $gr(\Pi)$ contains only a finite number of rules, and $\Delta_{i,h} \subseteq \Delta_{i,h+1}$ for all h, we must always reach some fixpoint Δ_i. That is, $\Delta_i = \Delta_{i,f}$ when $\Delta_{i,f} = \Delta_{i,f+1}$.

In order to establish our main result on computing the unique t-answer set (whenever one exists), we make use of the following lemma.

Lemma 2. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ. If I_1 and I_2 are t-models of $\mathcal{KB}^*_{\mu,i}$ for $i \geq 0$ such that $I_1 \cap \mathrm{HB}^*_{\mu,i-1}(\Pi) = I_2 \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)$ then $I_1 \cap I_2$ is a t-model of $\mathcal{KB}^*_{\mu,i}$.*

Intuitively, when we can extend a t-model of lower strata of the DL-program to a further stratum, there is always a subset minimal extension of this t-model.

By computing the t-answer set of $\mathcal{KB}^*_{\mu,0}$ and subsequently Δ_i for each stratum i, we can compute the t-answer set of $\mathcal{KB}$, whenever it exists:

Theorem 7. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ and let I be a t-answer set of $\mathcal{KB}^*_{\mu,i}$ for some $i > 0$. Then, $I = I'$ where $I' = (I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)) \cup \Delta_i$.*

Proof. Towards a contradiction assume $I \neq I'$. From Theorem 5 follows that $I' \notin \mathrm{AS}^{\mathrm{t}}(\mathcal{KB}^*_{\mu,i})$. As I is a minimal t-model of $\mathcal{KB}^*_{\mu,i}$, we get $I \cap I' \not\models_t^{\Phi} \Pi^*_{\mu,i}$. From this and Lemma 2 follows by modus tollens that $I' \not\models_t^{\Phi} \Pi^*_{\mu,i}$. Hence, there is a rule $r \in \Pi^*_{\mu,i}$ with $I' \models_t^{\Phi} B(r)$ and $I' \not\models_t^{\Phi} H(r)$. Consider the case that $\mu(H(r)) < i$. Then, $I' \not\models_t^{\Phi} r$ is a contradiction to $I \models_t^{\Phi} r$, since $I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi) = I' \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)$. Now consider case $\mu(H(r)) = i$ and number $m \leq 0$ such that $\Delta_{i,m} = \Delta_i$. As $\Delta_{i,m} \subseteq I'$ and $I' \not\models_t^{\Phi} H(r)$, we have $H(r) \notin \Delta_{i,m}$. Moreover, since $I' \models_t^{\Phi} B(r)$ and $I' = (I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)) \cup \Delta_{i,m}$, by Definition 10 we have that $H(r) \in \Delta_{i,m+1}$. As then $\Delta_{i,m} \neq \Delta_{i,m+1}$, we have a contradiction to $\Delta_{i,m}$ being the fixpoint Δ_i. □

So far we established that in case there is a t-answer set we can compute it stratum by stratum. In the following, we provide means for deciding the existence of a t-answer set during this computation.

Theorem 8. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ and I_{i-1} a t-answer set of $\mathcal{KB}^*_{\mu,i-1}$ for some $i > 0$. If $I_i = I_{i-1} \cup \Delta_i$ is a t-model of $\mathcal{KB}^*_{\mu,i}$ then I_i is a t-answer set of $\mathcal{KB}^*_{\mu,i}$.*

This enables us to pursue the following approach. After computing $I_i = I_{i-1} \cup \Delta_i$ for a stratum i, we check whether $I \models_t^{\Phi} \Pi^*_{\mu,i}$. If yes, we know by Theorem 8 that I_i is a t-answer set of $\mathcal{KB}^*_{\mu,i}$ and we are either done or continue our computation for stratum $i + 1$. If $I \not\models_t^{\Phi} \Pi^*_{\mu,i}$, we know by Theorem 7 that $\mathcal{KB}^*_{\mu,i}$ has no t-answer set and stop the computation.

Algorithm 1 for computing the t-answer set of a given DL-program $\mathcal{KB}$ with a t-stratification follows precisely this strategy after having computed the unique t-answer set of $\mathcal{KB}^*_{\mu,0}$. This can be done by a standard answer set solver as

Algorithm 1 Computing the t-answer set of a t-stratified DL-program $\mathcal{KB}$

Require: $\mathcal{KB} = (\Phi, \Pi)$, μ is a t-stratification of $\mathcal{KB}$ of length $k \geq 0$

1: $I_0 :=$ the unique t-answer set of $\mathcal{KB}^*_{\mu,0}$ // computable in exponential time
2: **for** $i := 1$ **to** k **do**
3: // compute Δ_i
4: $\Delta' := \emptyset$
5: **repeat**
6: $\Delta_i := \Delta'$
7: **for all** $r \in gr(\Pi_i)$ **do**
8: // loop may have exponentially many iterations
9: // the following check requires two queries to Φ per DL-atom in $B(r)$:
10: **if** $\Delta_i \cup I_{i-1} \models^{\Phi}_t B(r)$ **then**
11: $\Delta' := \Delta' \cup \{H(r)\}$
12: **end if**
13: **end for**
14: **until** $\Delta_i = \Delta'$ // number of iterations limited by number of rules in $gr(\Pi_i)$
15: $I_i := I_{i-1} \cup \Delta_i$
16: **if** $I_i \not\models^{\Phi}_t gr(\Pi^*_{\mu,i})$ **then**
17: **print** "$\mathcal{KB}$ has no t-answer set."
18: **return**
19: **end if**
20: **end for**
21: **return** I_k // I_k is the unique t-answer set of $\mathcal{KB}$

$\mathcal{KB}^*_{\mu,0}$ does not involve DL-atoms. Overall, the algorithm runs in exponential time with an additional effort of external calls to a DL-reasoner for evaluating the DL-queries of DL-atoms in lines 10 and 16. The time necessary for this evaluations depends on the complexity of query answering in the respective DL. Altogether, there may be an exponential number of such calls.

Theorem 9. *Given a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ, where query answering in Φ is in complexity class* C*, deciding whether $\mathcal{KB}$ has a t-answer set is in* ExpTime$^{\mathrm{C}}$.

When query answering in Φ is possible in exponential time, in the worst case the algorithm has to perform an exponential number of exponential time calls which can in turn be done in exponential time.

Theorem 10. *Given a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ, where query answering in Φ is in* ExpTime*, deciding whether $\mathcal{KB}$ has a t-answer set is* ExpTime*-complete.*

Hardness follows from ExpTime-completeness of ordinary stratified logic programs. For lightweight DLs such as those underlying OWL 2 EL, OWL 2 RL, and OWL 2 QL, where query answering has polynomial data complexity, reasoning for DL-programs is feasible in polynomial time under data complexity (where all of $\mathcal{KB}$ except facts in Φ and Π is fixed).

6 Conclusion and Outlook

We have introduced a refined semantics for DL-programs to overcome counter-intuitive results that are caused by inconsistency that emerges when combining rules and ontologies. For programs without DL-atoms our semantics coincides with the standard answer set semantics of logic programs. Moreover, we defined the property of t-stratification which guarantees that a DL-program has at most one answer set and gave an algorithm for computing it. Furthermore, we analyzed the computational complexity of the new semantics. The core of our approach is the definition of a new satisfaction relation for DL-rules such that for the body to be satisfied, additionally all its DL-atoms need to be DL-consistent.

An implementation of DL-inconsistency tolerant semantics is targeted in the context of the EU FP7 project OntoRule, with a focus on stratified programs and integration of F-Logic Programming [13].

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
2. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artificial Intelligence **172**(12-13) (2008) 1495–1539
3. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP'88, The MIT Press (1988) 1070–1080
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
5. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: IJCAI'05, Professional Book Center (2005) 90–96
6. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: JELIA'04. (2004)
7. Horrocks, I., Patel-Schneider, P.F.: Reducing OWL entailment to description logic satisfiability. In: ISWC'03. (2003)
8. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a Web ontology language. J. Web Sem. **1**(1) (2003) 7–26
9. Motik, B., Patel-Schneider, P.F., Parsia, B., eds.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. (2008) W3C Working Draft 02 December 2008.
10. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Well-founded semantics for description logic programs in the semantic web. In: RuleML'04. Volume 3323 of LNCS., Springer (2004) 81–97
11. Eiter, T., Ianni, G., Krennwallner, T., Schindlauer, R.: Exploiting Conjunctive Queries in Description Logic Programs. AMAI (1-4) (2008) 115–152
12. Krennwallner, T.: Integration of Conjunctive Queries over Description Logics into HEX-Programs. Master's thesis, Vienna University of Technology (2007)
13. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42**(4) (1995) 741–843