Artificial Intelligence 172 (2008) 1495–1539

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Combining answer set programming with description logics for the Semantic Web ☆

Thomas Eiter [a], Giovambattista Ianni [c,a,*], Thomas Lukasiewicz [b,a], Roman Schindlauer [a], Hans Tompits [a]

[a] *Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria*
[b] *Computing Laboratory, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK*
[c] *Dipartimento di Matematica, Università della Calabria, P.te P. Bucci, Cubo 30B, I-87036 Rende, Italy*

ARTICLE INFO

Article history:
Received 19 January 2007
Received in revised form 12 April 2008
Accepted 14 April 2008
Available online 25 April 2008

Keywords:
Answer set programming
Description logics
Rules
Ontologies
Semantic Web
Computational complexity
Closed-world reasoning
Default logic

ABSTRACT

We propose a combination of logic programming under the answer set semantics with the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which underly the Web ontology languages OWL Lite and OWL DL, respectively. To this end, we introduce *description logic programs* (or *dl-programs*), which consist of a description logic knowledge base L and a finite set P of *description logic rules* (or *dl-rules*). Such rules are similar to usual rules in nonmonotonic logic programs, but they may also contain *queries to* L, possibly under default negation, in their bodies. They allow for building rules on top of ontologies but also, to a limited extent, building ontologies on top of rules. We define a suite of semantics for various classes of dl-programs, which conservatively extend the standard semantics of the respective classes and coincide with it in absence of a description logic knowledge base. More concretely, we generalize positive, stratified, and arbitrary normal logic programs to dl-programs, and define a Herbrand model semantics for them. We show that they have similar properties as ordinary logic programs, and also provide fixpoint characterizations in terms of (iterated) consequence operators. For arbitrary dl-programs, we define answer sets by generalizing Gelfond and Lifschitz's notion of a transform, leading to a *strong* and a *weak answer set semantics*, which are based on reductions to the semantics of positive dl-programs and ordinary positive logic programs, respectively. We also show how the weak answer sets can be computed utilizing answer sets of ordinary normal logic programs. Furthermore, we show how some advanced reasoning tasks for the Semantic Web, including different forms of closed-world reasoning and default reasoning, as well as DL-safe rules, can be realized on top of dl-programs. Finally, we give a precise picture of the computational complexity of dl-programs, and we describe efficient algorithms and a prototype implementation of dl-programs which is available on the Web.

1. Introduction

The World Wide Web is impressively successful. Both the information that is stored on the Web and the number of its human users have been growing exponentially in recent years. For many people, the Web has started to play a fundamental role as a means of providing and searching for information. However, searching the Web in its current form is not always

☆ This paper is a significantly extended and revised version of a paper that appeared in: Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004), AAAI Press, 2004, pp. 141–151.

* Corresponding author.
E-mail addresses: eiter@kr.tuwien.ac.at (T. Eiter), ianni@kr.tuwien.ac.at (G. Ianni), lukasiewicz@kr.tuwien.ac.at (T. Lukasiewicz), roman@kr.tuwien.ac.at (R. Schindlauer), tompits@kr.tuwien.ac.at (H. Tompits).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2008.04.002

a joyful experience, since today's search engines often return a huge number of answers, many of which are completely irrelevant, while some relevant answers are not returned. One of the main reasons for this problem is that the current Web is designed for human consumption, but not for automated processing through machines, since the HTML standard only allows for describing the *layout* of Web pages, but not their *semantic content*.

The *Semantic Web* [9,10,36] is an extension of the current Web by standards and technologies that help machines to understand the information on the Web so that they can support richer discovery, data integration, navigation, and automation of tasks. The Semantic Web will not only allow for more exact answers when we search for information, but also provide knowledge necessary for integrating and comparing information from different sources, and allow for various forms of automated services. Roughly speaking, the main idea behind the Semantic Web is to add a machine-readable meaning to Web pages, to use ontologies for a precise definition of shared terms in Web resources, to make use of KR technology for automated reasoning from Web resources, and to apply cooperative agent technology for processing the information of the Web. The development of the Semantic Web proceeds in layers of Web technologies and standards, where every layer is lying on top of lower layers. The highest layer that has currently reached a sufficient maturity is the Ontology layer in the form of the *OWL Web Ontology Language* [58,101].

The language OWL provides three increasingly expressive sublanguages, namely *OWL Lite*, *OWL DL*, and *OWL Full*, where OWL DL basically corresponds to the Web ontology language DAML + OIL [52,53], which was developed by merging DAML [48] and OIL [35]. The languages OWL Lite and OWL DL are essentially very expressive description logics (DLs) with an RDF syntax [58]. One can therefore exploit a large body of existing previous work on description logic research, to define, for example, the formal semantics of the languages, to understand their formal properties (in particular, the decidability and the complexity of key inference problems), and for automated reasoning support. In fact, as shown by Horrocks and Patel-Schneider [54], ontology entailment in OWL Lite and OWL DL reduces to knowledge base (un)satisfiability in the expressive DLs $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, respectively.

The next step in the development of the Semantic Web is the realization of the Rules, Logic, and Proof layers, which are developed on top of the Ontology layer, and which should offer sophisticated representation and reasoning capabilities. A first effort in this direction was *RuleML* (Rule Markup Language) [11], fostering an XML-based markup language for rules and rule-based systems, while the OWL Rules Language [55] is a first proposal for extending OWL by Horn clause rules.

A key requirement of the layered architecture of the Semantic Web is to integrate the Rules and the Ontology layer. In particular, it is crucial to allow for building rules on top of ontologies, that is, for rule-based systems that use vocabulary specified in ontology knowledge bases. Another type of combination is to build ontologies on top of rules, which means that ontological definitions are supplemented by rules or imported from rules.

Towards the integration of rules and ontologies in the Semantic Web, we propose in this paper a combination of logic programming under the answer set semantics [39] with description logics, focusing here on the DLs $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which underly the Web ontology languages OWL Lite and OWL DL, respectively, as pointed out above. Our combination of dl-programs allows for building rules on top of ontologies, and also, to some extent, building ontologies on top of rules. Answer set semantics is the predominating semantics for nonmonotonic logic programs, and gave rise to the *answer set programming (ASP) paradigm* [7] in which the solutions for a problem are encoded in terms of the answer sets of a nonmonotonic logic program. Then, using an answer set solver, models (i.e., answer sets) of this program are generated, from which the solutions of the problem are read off. ASP has been successfully deployed to a variety of areas including diagnosis, configuration, planning, information integration, text mining, or security management, to name a few (cf. [105] for a comprehensive report about recent ASP applications).

The main innovations and contributions of this paper can be summarized as follows:

- We introduce *description logic programs* (or *dl-programs* for short), which consist of a knowledge base L in a description logic and a finite set P of *description logic rules* (or *dl-rules* for short). Such rules are similar to usual rules in logic programs with negation as failure, but they may also contain *queries to* L, possibly default negated, in their bodies. As an important feature, such queries also allow for specifying an input from P, and thus for a *flow of information from* P *to* L, besides the flow of information from L to P, given by any query to L. For example, concepts and roles in L may be enhanced by facts generated from dl-rules, possibly involving heuristic knowledge and other concepts and roles from L.
- Fostering an encapsulation view, the queries to L are treated in a way such that logic programming and DL inference are technically separated. Inspired by [25], merely interfacing details need to be known, while the components behind are black boxes. This approach, which provides a loose integration of rules and ontologies, is different from previous ones, which can be roughly divided into (i) hybrid approaches, which use DLs to specify structural constraints in the bodies of logic program rules, and (ii) approaches that reduce DL inference to logic programming. The basic idea behind (i) is to combine the semantic and computational strengths of the two different systems, while the main rationale of (ii) is to use powerful logic programming technology for inference in DLs. Both approaches differ significantly from our approach; this is discussed in detail in Section 9.
- We define a suite of semantics for various classes of dl-programs, which conservatively extend the standard semantics of the respective classes, and which coincide with it in absence of a DL knowledge base. More concretely, we generalize the classes of positive, stratified, and arbitrary normal logic programs to dl-programs, and define a Herbrand model semantics for them. We show that satisfiable positive dl-programs have a least Herbrand model, and that satisfiable stratified dl-programs can be associated with a unique minimal Herbrand model, which is characterized through a finite

number of iterative least Herbrand models. For arbitrary dl-programs, we define answer sets in the spirit of Gelfond and Lifschitz [39], for which we generalize their notion of a *transform*. We define the *strong answer set semantics*, which is based on a reduction to the least model semantics of positive dl-programs. We show that for positive and stratified dl-programs *KB*, the strong answer set semantics of *KB* coincides with the (unique) minimal Herbrand model semantics of *KB*. We also define the *weak answer set semantics* for general dl-programs, which is based on a reduction to the least model semantics of ordinary positive programs. Every strong answer set is also a weak answer set, but not vice versa. Both types of answer set semantics of general dl-programs properly generalize the answer set semantics of ordinary normal programs. In particular, the nondeterminism inherent in answer sets is retained, and the ASP problem solving paradigm thus extended to an integration of rules and ontologies.

- We give fixpoint characterizations for the unique minimal models of satisfiable positive and stratified dl-programs, and show how to compute them by fixpoint iteration and a sequence of finite fixpoint iterations, respectively. We also provide a general guess-and-check algorithm for computing the set of all weak answer sets of a general dl-program (which includes the set of all strong answer sets) by computing the set of all answer sets of an ordinary normal logic program. We also describe advanced algorithms which make use of these techniques, but also exploit structural properties like splitting sets and caching techniques for querying the DL knowledge base. They have been implemented in a working prototype implementation for dl-programs under the answer set semantics, which is available on the Web. To the best of our knowledge, it is currently the most advanced system for an integration of nonmonotonic rules and ontologies.
- We show that dl-programs under the answer set semantics can be fruitfully used to support advanced reasoning tasks for the Semantic Web. More concretely, we show that different forms of closed-world reasoning [40,41,88] and default reasoning [86,89] on top of DL knowledge bases can be elegantly realized using dl-programs. Furthermore, we show that dl-programs can be used to simulate DL-safe rules on DL knowledge bases [80].
- We give a precise picture of the complexity of deciding strong and weak answer set existence for a given dl-program, as well as of brave and cautious reasoning under both the weak and strong answer set semantics. We consider the general case as well as the restrictions where the given dl-program is positive or stratified. We consider the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, but most of our results can be easily transferred to other description logics of the same complexity (EXP resp. NEXP). In detail, for $KB = (L, P)$ with L in $\mathcal{SHIF}(\mathbf{D})$, answer set existence is EXP-complete if *KB* is positive or stratified, and NEXP-complete if *KB* is arbitrary. If L is in $\mathcal{SHOIN}(\mathbf{D})$, it is NEXP-complete if *KB* is positive, and $\mathrm{P}^{\mathrm{NEXP}}$-complete if *KB* is stratified or general. In nearly all cases, the complexity of cautious (resp., brave) reasoning from dl-programs coincides with the complexity of answer set non-existence (resp., existence) for dl-programs.

Our interfacing approach of dl-programs has several attractive features. First of all, it enables the usage of legacy software and solvers for answer set programs and DLs, respectively, to craft an engine for dl-programs. Furthermore, an engine for dl-programs will benefit from improvements to solvers for the components used. Another aspect is that the interfacing approach is amenable to distributed evaluation, and to privacy aspects for both the DL knowledge base L and the logic program P, since the internal structure of the one part need not be revealed to the other part for evaluation. This is particularly useful for realizing a service-oriented architecture of programs, in which access to an ontology is provided through a service.

The rest of this paper is organized as follows. Section 2 recalls normal logic programs under the answer set semantics, and Section 3 discusses the description logics $\mathcal{SHOIN}(\mathbf{D})$ and $\mathcal{SHIF}(\mathbf{D})$. In Section 4, we first introduce the syntax of dl-programs, and then define Herbrand models of dl-programs, unique minimal Herbrand models of positive and stratified dl-programs, and finally the strong and the weak answer set semantics for general dl-programs. Section 5 shows how the unique minimal Herbrand models of positive and stratified dl-programs can be computed through fixpoint iterations. It also gives a general guess-and-check algorithm for computing the set of all weak answer sets of general dl-programs. Section 6 shows how advanced reasoning tasks for the Semantic Web can be realized on top of dl-programs. In Section 7, we provide a precise picture of the complexity of deciding strong and weak answer set existence for a dl-program, and of brave and cautious reasoning from dl-programs under the weak and the strong answer set semantics. In Section 8, we describe advanced algorithms and a prototype implementation for dl-programs, and, in Section 9, we provide a detailed discussion on related work in the literature. Section 10 summarizes the main results and gives an outlook on further and future research. Detailed proofs of all results are relegated to Appendices A to F.

2. Normal programs under the answer set semantics

In this section, we recall normal programs (over classical literals) under the answer set semantics [39], which extends the stable model semantics [38] with *classical* (or, more appropriately, *strong*) *negation*. We first describe the syntax and then the semantics of normal logic programs. We finally describe some programming schemes.

2.1. Syntax

Let $\Phi = (\mathcal{P}, \mathcal{C})$ be a first-order vocabulary with nonempty finite sets $\mathcal{C}$ and $\mathcal{P}$ of constant resp. predicate symbols, but no function symbols. Let $\mathcal{X}$ be a set of variables. A *term* is either a variable from $\mathcal{X}$ or a constant symbol from Φ. An

atom is an expression of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity $n \geqslant 0$ from Φ, and $t_1, \ldots, t_n$ are terms. A *classical literal* (or simply *literal*) l is an atom p or a negated atom $\neg p$. Its *complementary* literal is $\neg p$ (resp., p). A *negation-as-failure literal* (or simply *NAF-literal*) is a literal l or a default-negated literal *notl*. A *normal rule* (or simply *rule*) r is an expression of the form

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

where a is a classical literal, and $b_1, \ldots, b_m$ are classical literals or *equality* (resp., *inequality*) atoms of the form $t_1 = t_2$ (resp., $t_1 \neq t_2$), where t_1 and t_2 are terms. The literal a is the *head* of the rule r, and the conjunction $b_1, \ldots, b_k$, *not* $b_{k+1}, \ldots, not\ b_m$ is the *body* of r, where $b_1, \ldots, b_k$ (resp., *not* $b_{k+1}, \ldots, not\ b_m$) is the *positive* (resp., *negative*) *body* of r. We use $H(r)$ to denote its head literal a, and $B(r)$ to denote the set of all its body literals $B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. If the body of the rule r is empty (that is, if $k = m = 0$), then r is a *fact*, and we often omit "$\leftarrow$" in such a case. A *normal program* (or simply *program*) P is a finite set of rules. A *positive program* P is a finite set of "*not*"-free rules.

2.2. Semantics

The answer set semantics of normal programs is defined in terms of consistent sets of classical literals, which represent three-valued interpretations. Positive programs are associated with their least satisfying consistent set of classical literals, if one exists, while the semantics of normal programs is defined by a reduction to the least model semantics of positive programs via the Gelfond–Lifschitz transformation.

More formally, the *Herbrand universe* of a program P, denoted HU_P, is the set of all constant symbols appearing in P. If there is no such constant symbol, then $HU_P = \{c\}$, where c is an arbitrary constant symbol from Φ. As usual, terms, atoms, literals, rules, programs, etc. are *ground* iff they do not contain any variables. The *Herbrand base* of a program P, denoted HB_P, is the set of all ground (classical) literals that can be constructed from the predicate symbols appearing in P and the constant symbols in HU_P. A *ground instance* of a rule $r \in P$ is obtained from r by replacing every variable that occurs in r by a constant symbol from HU_P, using a substitution θ for the variables in r, and removing all the valid equality and inequality atoms $t_1\theta = t_2\theta$ and $t_1\theta \neq t_2\theta$, respectively. A ground instance of r is *consistent* iff it contains no equality or inequality atoms. We denote by $ground(P)$ the set of all consistent ground instances of rules in P.

A set $X \subseteq HB_P$ of literals is *consistent* iff $\{p, \neg p\} \not\subseteq X$ for every atom $p \in HB_P$. An *interpretation* I relative to a program P is a consistent subset of HB_P. Intuitively, any such I represents a three-valued interpretation of all ground atoms as follows: an atom a has the truth value *true*, *false*, and *unknown* iff $a \in I$, $\neg a \in I$, and $\{a, \neg a\} \cap I = \emptyset$, respectively. A *model* of a positive program P is an interpretation $I \subseteq HB_P$ such that $B(r) \subseteq I$ implies $H(r) \in I$, for every $r \in ground(P)$. An *answer set* of a positive program P is the least model of P with respect to set inclusion.

The *transform*, or *Gelfond–Lifschitz transform*, of a program P relative to an interpretation $I \subseteq HB_P$, denoted P^I, is the ground positive program that is obtained from $ground(P)$ by (i) deleting every rule r such that $B^-(r) \cap I \neq \emptyset$, and (ii) deleting the negative body from every remaining rule. An *answer set* of a (normal) program P is an interpretation $I \subseteq HB_P$ such that I is an answer set of P^I. The set of all answer sets of a program P is denoted by $ans(P)$.

The main reasoning tasks for programs under the answer set semantics are the following:

(1) decide whether a given program P has an answer set;
(2) given a program P and a ground literal l, decide whether l is in every (resp., some) answer set of P, denoted $P \models_c l$ (resp., $P \models_b l$), called *cautious* (resp., *brave*) *reasoning*;
(3) given a program P and an interpretation $I \subseteq HB_P$, decide whether I is an answer set of P, called *answer set checking*; and
(4) compute the set $ans(P)$ of all answer sets of a given program P.

The following two examples describe common programming schemes for normal programs under the answer set semantics, which will be used in the sequel. Variants of the schema illustrated by the first example are adopted for enforcing multiple answers to a given normal program, so that a variety of types of nondeterministic choice can be modeled.

Example 2.1 *(Guess module).* Consider the normal logic program P, consisting of the following rules, where g is an atom:

$$g \leftarrow not\ \neg g; \qquad \neg g \leftarrow not\ g. \tag{2}$$

Then, the answer sets of P are given by $M_1 = \{g\}$ and $M_2 = \{\neg g\}$.

The next example shows another common schema, which allows to enforce the inconsistency of an answer set depending on some given condition.

Example 2.2 *(Constraint).* Let the normal program P be obtained from a given normal program by adding the following rule, where f is a fresh atom:

$$f \leftarrow c, not\ f. \tag{3}$$

Then, no answer set of P contains c. Indeed, in every such answer set M, $f \leftarrow not\ f$ would have to hold; this, however, is not possible under answer set semantics, since absence of f in M would enforce presence of f in M and vice versa.

3. The description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$

In this section, we recall the syntax and the semantics of the expressive description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which provide the logical underpinning of the Web ontology languages OWL Lite and OWL DL, respectively (see [54,58] for further details and background). Intuitively, description logics model a domain of interest in terms of concepts and roles, which represent classes of individuals and binary relations on classes of individuals, respectively. A description logic knowledge base encodes in particular subset relationships between classes of individuals, subset relationships between binary relations on classes of individuals, the membership of individuals to classes, and the membership of pairs of individuals to binary relations on classes. Other important ingredients of $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$) are datatypes (resp., datatypes and individuals) in concept expressions.

3.1. Syntax

We now recall the syntax of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. We first describe the syntax of the latter, which has the following datatypes and elementary ingredients. We assume a set $\mathbf{E}$ of *elementary datatypes* and a set $\mathbf{V}$ of *data values*. A *datatype theory* $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a *datatype* (or *concrete*) *domain* $\Delta^{\mathbf{D}}$ and a mapping $\cdot^{\mathbf{D}}$ that assigns to every elementary datatype a subset of $\Delta^{\mathbf{D}}$ and to every data value an element of $\Delta^{\mathbf{D}}$. Let $\Psi = (\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D, \mathbf{I} \cup \mathbf{V})$ be a vocabulary, where $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ are pairwise disjoint (denumerable) sets of *atomic concepts*, *abstract roles*, *datatype* (or *concrete*) *roles*, and *individuals*, respectively. We denote by $\mathbf{R}_A^-$ the set of inverses R^- of all $R \in \mathbf{R}_A$.

Roles and concepts are defined as follows. A *role* is an element of $\mathbf{R}_A \cup \mathbf{R}_A^- \cup \mathbf{R}_D$. *Concepts* are inductively defined as follows. Every atomic concept $C \in \mathbf{A}$ is a concept. If $o_1, o_2, \ldots$ are individuals from $\mathbf{I}$, then $\{o_1, o_2, \ldots\}$ is a concept (called *oneOf*). If C and D are concepts, then also $(C \sqcap D)$, $(C \sqcup D)$, and $\neg C$ are concepts (called *conjunction*, *disjunction*, and *negation*, respectively). If C is a concept, R is an abstract role from $\mathbf{R}_A \cup \mathbf{R}_A^-$, and n is a nonnegative integer, then $\exists R.C$, $\forall R.C$, $\geqslant nR$, and $\leqslant nR$ are concepts (called *exists*, *value*, *atleast*, and *atmost restriction*, respectively). If D is a datatype, U is a datatype role from $\mathbf{R}_D$, and n is a nonnegative integer, then $\exists U.D$, $\forall U.D$, $\geqslant nU$, and $\leqslant nU$ are concepts (called *datatype exists*, *value*, *atleast*, and *atmost restriction*, respectively). We use $\top$ and $\bot$ to abbreviate the concepts $C \sqcup \neg C$ and $C \sqcap \neg C$, respectively, and we eliminate parentheses as usual.

We next define axioms and knowledge bases as follows. An *axiom* is an expression of one of the following forms:

(1) $C \sqsubseteq D$, called *concept inclusion axiom*, where C and D are concepts;
(2) $R \sqsubseteq S$, called *role inclusion axiom*, where either $R, S \in \mathbf{R}_A$ or $R, S \in \mathbf{R}_D$;
(3) $\mathrm{Trans}(R)$, called *transitivity axiom*, where $R \in \mathbf{R}_A$;
(4) $C(a)$, called *concept membership axiom*, where C is a concept and $a \in \mathbf{I}$;
(5) $R(a, b)$ (resp., $U(a, v)$), called *role membership axiom*, where $R \in \mathbf{R}_A$ (resp., $U \in \mathbf{R}_D$) and $a, b \in \mathbf{I}$ (resp., $a \in \mathbf{I}$ and v is a data value); and
(6) $a = b$ (resp., $a \neq b$), or $= (a, b)$ (resp., $\neq (a, b)$), called *equality* (resp., *inequality*) *axiom*, where $a, b \in \mathbf{I}$.

We also use $F \equiv G$ to abbreviate the two concept or role inclusion axioms $F \sqsubseteq G$ and $G \sqsubseteq F$. A *description logic (DL) knowledge base* L is a finite set of axioms.

For an abstract role $R \in \mathbf{R}_A$, we define $\mathrm{Inv}(R) = R^-$ and $\mathrm{Inv}(R^-) = R$. Let the *transitive and reflexive closure of* $\sqsubseteq$ on abstract roles *relative to* L, denoted $\sqsubseteq^\star$, be defined as follows: For two abstract roles R and S in L, $S \sqsubseteq^\star R$ relative to L holds iff either (a) $S = R$, (b) $S \sqsubseteq R \in L$, (c) $\mathrm{Inv}(S) \sqsubseteq \mathrm{Inv}(R) \in L$, or (d) some abstract role Q exists such that $S \sqsubseteq^\star Q$ and $Q \sqsubseteq^\star R$ relative to L. An abstract role R is *simple relative to* L iff, for each abstract role S such that $S \sqsubseteq^\star R$ relative to L, it holds that (i) $\mathrm{Trans}(S) \notin L$ and (ii) $\mathrm{Trans}(\mathrm{Inv}(S)) \notin L$. For decidability, number restrictions in L are restricted to simple abstract roles [59].

Observe that in $\mathcal{SHOIN}(\mathbf{D})$, concept and role membership axioms can also be expressed through concept inclusion axioms. The knowledge that the individual a is an instance of the concept C can be expressed by the concept inclusion axiom $\{a\} \sqsubseteq C$, while the knowledge that the pair (a, b) (resp., (a, v)) is an instance of the role R (resp., U) can be expressed by $\{a\} \sqsubseteq \exists R.\{b\}$ (resp., $\{a\} \sqsubseteq \exists U.\{v\}$).

The syntax of $\mathcal{SHIF}(\mathbf{D})$ is the one of $\mathcal{SHOIN}(\mathbf{D})$, but without the oneOf constructor and with the atleast and atmost constructors limited to 0 and 1.

The following example introduces a DL knowledge base L_S for a reviewer selection scenario, which is also used in some subsequent examples.

Example 3.1 *(Reviewer selection).* Suppose we want to assign reviewers to papers, based on certain information about papers and available persons, which is encoded in the DL knowledge base L_S. More concretely, L_S classifies papers into research areas. A research area belongs to the concept *Area*. A *paper* is classified depending on keyword information. The abstract roles *keyword* and *inArea* associate with each paper its relevant keywords and the areas that it is classified into, respectively.

A paper is in an area if it is associated with a keyword of that area. The abstract role *expert* relates persons to their areas of expertise. For simplicity, a person is an expert in an area if he or she wrote a paper in that area. The concept *Referee* contains all referees. The abstract role *contains* associates each area with a group of keywords, while *hasMember* relates clusters of keywords having some overlap (e.g., one could put in the same cluster C_1 the words "Semantic Web" and "Ontologies"). The following are some axioms from L_S:

$$\mathit{Paper} \sqsubseteq \mathit{Publication}; \quad \mathit{Referee} \sqsubseteq \mathit{Person};$$

$$\mathit{Paper}(\mathit{pub}_1); \quad \mathit{Referee}(\mathit{per}_1); \quad \mathit{Referee}(\mathit{per}_2);$$

$$\mathit{hasMember}(C_1, \mathit{SemanticWeb}); \quad \mathit{Author}(\mathit{pub}_1, \mathit{per}_2);$$

$$\mathit{Area}(A); \quad \mathit{Area}(B); \quad \mathit{Area}(C); \quad \mathit{Area}(D); \quad \mathit{Area}(E);$$

$$\mathit{contains}(A, \mathit{Agents}); \quad \mathit{keyword}(\mathit{pub}_1, \mathit{Agents});$$

$$\exists \mathit{inArea}.\{\mathrm{c}\} \equiv \exists \mathit{keyword}.(\exists \mathit{contains}^{-}.\{\mathrm{c}\}), \quad \text{for all } \mathrm{c} \in \{A, B, C, D, E\};$$

$$\exists \mathit{expert}.\{\mathrm{c}\} \equiv \exists \mathit{Author}^{-}.(\exists \mathit{inArea}.\{\mathrm{c}\}), \quad \text{for all } \mathrm{c} \in \{A, B, C, D, E\}.$$

3.2. Semantics

We now define the semantics of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ in terms of general first-order interpretations, as usual, and we also recall some important reasoning problems in description logics.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to a datatype theory $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a nonempty (*abstract*) *domain* $\Delta^{\mathcal{I}}$ disjoint from $\Delta^{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$ that assigns to each atomic concept $C \in \mathbf{A}$ a subset of $\Delta^{\mathcal{I}}$, to each individual $o \in \mathbf{I}$ an element of $\Delta^{\mathcal{I}}$, to each abstract role $R \in \mathbf{R}_A$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each datatype role $U \in \mathbf{R}_D$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}}$. The mapping $\cdot^{\mathcal{I}}$ is extended to all concepts and roles as usual (where $\#S$ denotes the cardinality of a set S):

- $(R^{-})^{\mathcal{I}} = \{(a, b) \mid (b, a) \in R^{\mathcal{I}}\}$;
- $\{o_1, \ldots, o_n\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\}$;
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, and $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$;
- $(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$;
- $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$;
- $(\geqslant nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in R^{\mathcal{I}}\}) \geqslant n\}$;
- $(\leqslant nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in R^{\mathcal{I}}\}) \leqslant n\}$;
- $(\exists U.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x, y) \in U^{\mathcal{I}} \wedge y \in D^{\mathbf{D}}\}$;
- $(\forall U.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x, y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathbf{D}}\}$;
- $(\geqslant nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in U^{\mathcal{I}}\}) \geqslant n\}$;
- $(\leqslant nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x, y) \in U^{\mathcal{I}}\}) \leqslant n\}$.

The *satisfaction* of a description logic axiom F in the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$, denoted $\mathcal{I} \models F$, is defined as follows: (1) $\mathcal{I} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; (2) $\mathcal{I} \models R \sqsubseteq S$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$; (3) $\mathcal{I} \models \mathrm{Trans}(R)$ iff $R^{\mathcal{I}}$ is transitive; (4) $\mathcal{I} \models C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$; (5) $\mathcal{I} \models R(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ (resp., $\mathcal{I} \models U(a, v)$ iff $(a^{\mathcal{I}}, v^{\mathbf{D}}) \in U^{\mathcal{I}}$); and (6) $\mathcal{I} \models a = b$ iff $a^{\mathcal{I}} = b^{\mathcal{I}}$ (resp., $\mathcal{I} \models a \neq b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$). The interpretation $\mathcal{I}$ *satisfies* the axiom F, or $\mathcal{I}$ is a *model* of F, iff $\mathcal{I} \models F$. The interpretation $\mathcal{I}$ *satisfies* a DL knowledge base L, or $\mathcal{I}$ is a *model* of L, denoted $\mathcal{I} \models L$, iff $\mathcal{I} \models F$ for all $F \in L$. We say that L is *satisfiable* (resp., *unsatisfiable*) iff L has a (resp., no) model. An axiom F is a *logical consequence* of L, denoted $L \models F$, iff every model of L satisfies F. A negated axiom $\neg F$ is a *logical consequence* of L, denoted $L \models \neg F$, iff every model of L does not satisfy F.

Some important reasoning problems related to DL knowledge base L are the following: (1) decide whether a given L is satisfiable; (2) given L and a concept C, decide whether $L \not\models C \sqsubseteq \bot$; (3) given L and two concepts C and D, decide whether $L \models C \sqsubseteq D$; (4) given L, an individual $a \in \mathbf{I}$, and a concept C, decide whether $L \models C(a)$; (5) given L, two individuals $a, b \in \mathbf{I}$ (resp., an individual $a \in \mathbf{I}$ and a data value v), and an abstract role $R \in \mathbf{R}_A$ (resp., a datatype role $U \in \mathbf{R}_D$), decide whether $L \models R(a, b)$ (resp., $L \models U(a, v)$), and (6) given L and two individuals $a, b \in \mathbf{I}$, decide whether $L \models a = b$ or whether $L \models a \neq b$.

Here, (1) is a special case of (2), since L is satisfiable iff $L \not\models \top \sqsubseteq \bot$. Furthermore, (2) and (3) can be reduced to each other, since $L \models C \sqcap \neg D \sqsubseteq \bot$ iff $L \models C \sqsubseteq D$. Finally, in $\mathcal{SHOIN}(\mathbf{D})$, since concept and role membership axioms can also be expressed through concept inclusion axioms (see above), (4) and (5) are special cases of (3).

Example 3.2 *(Reviewer selection, ctd.).* It is not difficult to verify that the set L of all axioms given in Example 3.1 is satisfiable, and thus that the class of all papers and the class of all publications both may have some instances. Furthermore, L logically implies the axiom $\mathit{inArea}(\mathit{pub}_1, \mathit{Agents})$, which follows from the axioms (5) to (6) in Example 3.1.

4. Description logic programs

In this section, we introduce *description logic programs* (or simply *dl-programs*), which are a novel combination of normal programs under the answer set semantics and DL knowledge bases under their standard first-order semantics. We first define the syntax of dl-programs and then their semantics. We finally discuss some further semantic properties of dl-programs.

4.1. Syntax

Informally, a dl-program consists of a DL knowledge base L and a generalized normal program P. The latter is a finite set of generalized rules, which may contain queries to L in their body. In such a query, it is asked whether a certain description logic axiom or its negation logically follows from L.

The DL knowledge base L is defined over a vocabulary $\Psi = (\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D, \mathbf{I} \cup \mathbf{V})$, as in Section 3.1, while the generalized normal program P is defined over a vocabulary $\Phi = (\mathcal{P}, \mathcal{C})$, as in Section 2.1. We assume that $\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D$ is disjoint from $\mathcal{P}$, while $I_P \subseteq \mathcal{C} \subseteq \mathbf{I} \cup \mathbf{V}$, where I_P is the set of all constant symbols appearing in P. Note that $\mathcal{C}$ does not necessarily contain all the named individuals explicitly appearing in L; it may be arbitrarily chosen, although a primary setting (which is the default behavior of our system prototype) is choosing $\mathcal{C}$ such that $I_L \subseteq \mathcal{C}$, where I_L is the set of all individuals appearing in L.[1]

We now define the notions of dl-queries and dl-atoms, which are used in rule bodies to express queries to the DL knowledge base L. A *dl-query* $Q(\mathbf{t})$ is either (a) a concept inclusion axiom F or its negation $\neg F$; (b) of the forms $C(t)$ or $\neg C(t)$, where C is a concept, and t is a term; (c) of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role, and t_1 and t_2 are terms; or (d) of the forms $= (t_1, t_2)$ and $\neq (t_1, t_2)$, where t_1 and t_2 are terms. Note here that $\mathbf{t}$ is the empty argument list in (a), $\mathbf{t} = t$ in (b), and $\mathbf{t} = (t_1, t_2)$ in (c) and (d), and terms are defined in the same way as in Section 2.1. A *dl-atom* has the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t}), \quad m \geqslant 0, \tag{4}$$

where each S_i is either a concept, a role, or a special symbol $\theta \in \{=, \neq\}$; $op_i \in \{\uplus, \cup, \cap\}$; p_i is a unary predicate symbol, if S_i is a concept, and a binary predicate symbol otherwise; and $Q(\mathbf{t})$ is a dl-query. We call $p_1, \ldots, p_m$ its *input predicate symbols*. 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* $KB = (L, P)$ consists of a description logic knowledge base L and a finite set of dl-rules P.

4.2. Semantics

Prior to defining the semantics of dl-programs, we first give an intuitive outline via an example. We then introduce Herbrand models of dl-programs and, in analogy to the development for ordinary logic programs, gradually introduce genuine semantics for increasingly more expressive fragments of dl-programs: we begin with a canonical least model semantics for positive dl-programs, and then define a canonical iterative least model semantics for stratified dl-programs. Finally, we define for arbitrary dl-programs two alternative notions of answer sets, namely, *strong* and *weak answer sets*. Note that (consistent) positive and stratified dl-programs both have unique single answer sets, while non-stratified dl-programs may have multiple answer sets.

4.2.1. Illustrating example

Roughly speaking, dl-programs are a means for coupling two knowledge sources, namely, a DL knowledge base and a logic program, while assuring the possibility for the two sources to exchange information.

Example 4.1 *(Reviewer selection, ctd.).* Let $KB_S = (L_S, P_S)$ be the dl-program consisting of the DL knowledge base L_S from Example 3.1 and the set P_S, given by the following dl-rules:

$$paper(p_1);\ \ kw(p_1, Semantic\_Web); \tag{5}$$

$$paper(p_2);\ \ kw(p_2, Bioinformatics); \tag{6}$$

$$kw(p_2, Answer\_Set\_Programming); \tag{7}$$

$$kw(P, K_2) \leftarrow kw(P, K_1),\ \ DL[hasMember](S, K_1),\ \ DL[hasMember](S, K_2); \tag{8}$$

$$paperArea(P, A) \leftarrow DL[keywords \uplus kw; inArea](P, A); \tag{9}$$

$$cand(X, P) \leftarrow paperArea(P, A),\ \ DL[Referee](X),\ \ DL[expert](X, A); \tag{10}$$

[1] This allows for information hiding, as only selected individuals occurring in L might be shown to the user.

$$assign(X, P) \leftarrow cand(X, P), not\ \neg assign(X, P); \tag{11}$$

$$\neg assign(Y, P) \leftarrow cand(Y, P), assign(X, P), X \neq Y; \tag{12}$$

$$a(P) \leftarrow assign(X, P); \tag{13}$$

$$error(P) \leftarrow paper(P), not\ a(P). \tag{14}$$

The formal semantics, defined subsequently, associates KB_S with a collection of *answer sets*, like for the case of ordinary logic programs. The purpose of P_S is to specify how these answer sets should look like.

Facts (5) to (7) specify two papers, p_1 and p_2, to be assigned to reviewers along with their keywords. Rule (8) allows for retrieving keyword information from L_S. More concretely, the predicate kw is augmented via dl-atoms by those keywords in L_S that share the same area. Intuitively, the ground dl-atom $DL[hasMember](s, k)$ is true for all pairs (s, k) such that $L_S \models hasMember(s, k)$. Rule (9) contains a richer kind of dl-atom, where we first enrich the role *keywords* in L_S by the extension of the predicate kw in P_S via the operator $\uplus$. We then query the role *inArea* over the modified version L'_S of L_S (i.e., we retrieve from L'_S the areas that each paper is classified into). Here, the new information coming from kw might trigger new information available in L'_S.

In Rule (10), we define reviewer candidates for a given paper. More specifically, a reviewer x is a candidate to review the paper p, if x is known in L_S as a *Referee* and as an *expert* in the reference area a of p. Rules (11) and (12) encode a nondeterministic choice. As we will see, our semantics gives a special meaning to rules that appear in recursive rules and involve negation. Intuitively, for any possible candidate pair $cand(x, p)$, we guess nondeterministically whether $assign(x, p)$ is true or not, meaning that p must be assigned to x for reviewing or not. Thus, there are answers where $assign(x, p)$ is true, and there are other answers where it is false. Finally, Rules (13) and (14) check if each paper is assigned; if not, then an error is flagged: answers where $error(p)$ is true for a given p can be filtered out.

Note that Rules (8) to (10) transfer knowledge from L_S to P_S. In Rule (9), knowledge is also transferred from P_S to L_S. Hence, information flows in both directions between the DL knowledge base L_S and the generalized program P_S.

The intuitive meaning of the operator $\uplus$ is to add information from P_S as negative assertions. For instance, $keywords \uplus kw$ means that L_S is enlarged with assertions $\neg keywords(k)$ for any true $kw(k)$. To illustrate the use of $\cap$, imagine to define a unary predicate *poss_Referees* in the dl-program, and to add $Referee \cap poss\_Referees$ in the first dl-atom of (10). The effect of this modification would be to add to L_S negative assertions $\neg Referee(r)$ for all the r such that *poss_Referees does not* hold, thus constraining possible referees to the domain of *poss_Referees*.

4.2.2. Models of dl-programs

We first define Herbrand interpretations and satisfaction of dl-programs in Herbrand interpretations. The latter hinges on defining the truth of ground dl-atoms in Herbrand interpretations. In what follows, let $KB = (L, P)$ be a dl-program over the vocabulary $\Phi = (\mathcal{P}, \mathcal{C})$.

The *Herbrand base* of P, denoted HB_P, is the set of all ground literals built with (a) predicate symbols in $\mathcal{P}$ that occur in P and (b) constant symbols in $\mathcal{C}$. An *interpretation* I relative to P is a consistent subset of HB_P. Such an I is a *model* of a ground literal or dl-atom l (or I *satisfies* l) under L, denoted $I \models_L l$, if the following holds:

- if $l \in HB_P$, then $I \models_L l$ iff $l \in I$;
- if l is a ground dl-atom $DL[\lambda; Q](\mathbf{c})$, where $\lambda = S_1 op_1\ p_1, \ldots, S_m op_m p_m$, then $I \models_L l$ iff $L(I; \lambda) \models Q(\mathbf{c})$, where $L(I; \lambda) = L \cup \bigcup_{i=1}^{m} A_i(I)$ and, for $1 \leqslant i \leqslant m$,

$$A_i(I) = \begin{cases} \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \uplus; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \cup; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}, & \text{if } op_i = \cap. \end{cases}$$

We say that I is a *model* of a ground dl-rule r iff $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$ implies $I \models_L H(r)$. We say I is a *model* of a dl-program $KB = (L, P)$, or I *satisfies* KB, denoted $I \models KB$, iff $I \models_L r$ for all $r \in ground(P)$. We say KB is *satisfiable* (resp., *unsatisfiable*) iff it has some (resp., no) model.

Observe that the above satisfaction of dl-atoms a in interpretations I also involves negated concept inclusion axioms $\neg(C \sqsubseteq D)$, negated concept membership axioms $\neg C(a)$, and negated role membership axioms $\neg R(a, b)$ and $\neg U(a, v)$. For this reason, we slightly extend the standard syntax and semantics of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ by also allowing such negated axioms.[2] The notions of satisfaction, satisfiability, and entailment are naturally extended to handle such axioms. In particular, an interpretation $\mathcal{I}$ satisfies $\neg(C \sqsubseteq D)$ (resp., $\neg C(a)$, $\neg R(a, b)$, $\neg U(a, v)$) iff $C^{\mathcal{I}} \not\subseteq D^{\mathcal{I}}$ (resp., $a^{\mathcal{I}} \notin C^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}$, $(a^{\mathcal{I}}, v^{\mathbf{D}}) \notin U^{\mathcal{I}}$).

Entailment (for dl-atoms) in the slight extensions of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ can be reduced to entailment in $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, respectively, as follows. Notice first that the entailment of a concept inclusion, concept membership, role membership, or equality axiom F (resp., its negation $\neg F$) from a DL knowledge base L is equivalent to the

[2] Actually, OWL 2 follows a similar pattern, allowing for negative property membership assertions [102].

unsatisfiability of $L \cup \{\neg F\}$ (resp., $L \cup \{F\}$). Here, the negated concept inclusion axiom $\neg(C \sqsubseteq D)$ is equivalent to the concept membership axiom $(C \sqcap \neg D)(a)$ (where a is a fresh individual), and the negated concept membership axiom $\neg(C(a))$ is equivalent to the concept membership axiom $(\neg C)(a)$. Then, every negated abstract role membership axiom in a DL knowledge base L can be removed by using that $L' \cup \{\neg R(a,b)\}$ is unsatisfiable iff $L' \cup \{A(a),\ B(b),\ \exists R.B \sqsubseteq \neg A\}$ is unsatisfiable (where A and B are two fresh atomic concepts and L' is any DL knowledge base) [72]. Negated datatype role membership axioms can be removed in a similar way.

4.2.3. Least model semantics of positive dl-programs

We now define positive dl-programs, which are informally dl-programs that contain no default negations and that involve only monotonic dl-atoms. Like ordinary positive programs, every positive dl-program that is satisfiable has a unique least model, which naturally characterizes its semantics.

We first define the notion of monotonicity for dl-atoms as follows. A ground dl-atom a is *monotonic* relative to $KB = (L, P)$ iff $I \models_L a$ implies $I' \models_L a$, for all $I \subseteq I' \subseteq HB_P$, otherwise a is *nonmonotonic*. Observe that a dl-atom containing the operator $\cap$ may fail to be monotonic, since an increasing set of $p_i(\mathbf{e})$ in P results in a reduction of $\neg S_i(\mathbf{e})$ in L, whereas dl-atoms containing only the operators $\uplus$ and $\cup$ are always monotonic. A dl-program $KB = (L, P)$ is *positive* iff (i) P is "*not*"-free, and (ii) every ground dl-atom that occurs in *ground*(P) is monotonic relative to KB.

For ordinary positive programs P, the intersection of two models of P is also a model of P. The following lemma shows a similar result for positive dl-programs.

Lemma 4.2. *Let $KB = (L, P)$ be a positive dl-program. If the interpretations $I_1, I_2 \subseteq HB_P$ are models of KB, then $I_1 \cap I_2$ is also a model of KB.*

An immediate corollary is the following proposition.

Corollary 4.3. *Let $KB = (L, P)$ be a positive dl-program. If KB is satisfiable, then there exists a unique model $I \subseteq HB_P$ of KB such that $I \subseteq J$ for all models $J \subseteq HB_P$ of KB, i.e., a unique least model of KB.*

In the spirit of Logic Programming, this special model is adopted as the semantics of KB.

Definition 4.4. For every satisfiable positive dl-program $KB = (L, P)$, we denote by M_{KB} its unique least model.

Example 4.5. Let KB result from the dl-program KB_S of Example 4.1 by removing Rules (11)–(14). Clearly, KB is "*not*"-free. Moreover, since the dl-atoms do not contain $\cap$, they are all monotonic. Thus, KB is positive. As well, its unique least model M_{KB} contains all review candidates for the given papers p_1 and p_2.

4.2.4. Iterative least model semantics of stratified dl-programs

We next define stratified dl-programs, which are intuitively composed of hierarchic layers of positive dl-programs. In the traditional notion of stratification, it is required that the head-body dependency enforces a partial order between ground atoms, where default-negated ground body atoms must strictly precede their ground head atom in such an order. This condition is in the following extended to ground dl-atoms not known to be monotonic. Like for ordinary stratified programs, a canonical minimal model of stratified dl-programs can be singled out by a number of iterative least models, which naturally describes the semantics, provided some model exists. We can accommodate this with possibly nonmonotonic dl-atoms by treating them similarly as NAF-literals. This is particularly useful, because in general it is not known a priori whether a given dl-atom is monotonic, and determining this might be costly; notice, however, that absence of $\cap$ in (4) is a simple syntactic criterion that implies monotonicity of a dl-atom (cf. also Example 4.5).

For any dl-program $KB = (L, P)$, we denote by DL_P the set of all ground dl-atoms that occur in *ground*(P). We assume that KB has an associated set $DL_P^+ \subseteq DL_P$ of ground dl-atoms which are known to be monotonic, and we denote by $DL_P^? = DL_P \setminus DL_P^+$ the set of all other ground dl-atoms. An *input literal* of some dl-atom $a \in DL_P$ is a ground literal with an input predicate of a and constant symbols in Φ.

The notion of a stratification for dl-programs defines an ordered partition of the set of all ground atoms and ground dl-atoms as follows.

Definition 4.6. Let $KB = (L, P)$ be a dl-program. A *stratification of KB* (*relative to* DL_P^+) is a mapping $\mu : HB_P \cup DL_P \to \{0, 1, \ldots, k\}$ such that

(i) for each $r \in ground(P)$, $\mu(H(r)) \geqslant \mu(l')$ for each $l' \in B^+(r)$, and $\mu(H(r)) > \mu(l')$ for each $l' \in B^-(r)$, and
(ii) $\mu(a) \geqslant \mu(l)$ for each input literal l of each $a \in DL_P^+$, and $\mu(a) > \mu(l)$ for each input literal l of each $a \in DL_P^?$.

We call $k \geqslant 0$ the *length* of μ. For every $i \in \{0, \ldots, k\}$, we then define the dl-program KB_i as $(L, P_i) = (L, \{r \in ground(P) \mid \mu(H(r)) = i\})$, and HB_{P_i} (resp., $HB^*_{P_i}$) as the set of all $l \in HB_P$ such that $\mu(l) = i$ (resp., $\mu(l) \leqslant i$).

We say that a dl-program $KB = (L, P)$ is *stratified*, iff it has a stratification μ of some length $k \geqslant 0$. Its canonical model is determined as follows.

Definition 4.7. Let $KB = (L, P)$ be a dl-program with a stratification of length $k \geqslant 0$. We define its iterative least models $M_i \subseteq HB_P$ with $i \in \{0, \ldots, k\}$ by:

(i) M_0 is the least model of KB_0;
(ii) if $i > 0$, then M_i is the least subset M of HB_P such that (a) M is a model of KB_i and (b) $M \cap HB^{\star}_{P_{i-1}} = M_{i-1} \cap HB^{\star}_{P_{i-1}}$.

We call KB *consistent*, if every M_i with $i \in \{0, \ldots, k\}$ exists, and *inconsistent* otherwise. If KB is consistent, then M_{KB} denotes the canonical model M_k.

Note that M_{KB} is well-defined, as it does not dependent on a particular μ (as also witnessed by Corollary 4.15). The following result shows that M_{KB} is in fact a minimal model of KB.

Theorem 4.8. *Let $KB = (L, P)$ be a stratified dl-program. Then, M_{KB} is a minimal model of KB.*

Example 4.9. Let KB be the dl-program resulting from the dl-program KB_S of Example 4.1 by removing Rules (11) and (12). This program has a stratification of length 2, with the associated set DL^+_P comprising all dl-atoms occurring in P. The least model of P contains all review candidates of the given papers, together with error flags for them, because no paper is assigned so far.

4.2.5. Strong answer set semantics of dl-programs

We now define the *strong answer set semantics* of general dl-programs, which is reduced to the least model semantics of positive dl-programs. We use a generalized transformation that removes all NAF-literals and all dl-atoms except for those known to be monotonic. If we ignore this knowledge and remove all dl-atoms, then we arrive at the *weak answer set semantics* (see Section 4.2.6).

Definition 4.10. Let $KB = (L, P)$ be a dl-program and let DL_P, DL^+_P, and $DL^?_P$ be as above. The *strong dl-transform* of P relative to L and an interpretation $I \subseteq HB_P$, denoted sP^I_L, is the set of all dl-rules obtained from $ground(P)$ by deleting

(i) every dl-rule r such that either $I \not\models_L a$ for some $a \in B^+(r) \cap DL^?_P$, or $I \models_L l$ for some $l \in B^-(r)$; and
(ii) from each remaining dl-rule r all literals in $B^-(r) \cup (B^+(r) \cap DL^?_P)$.

Notice that (L, sP^I_L) has only monotonic dl-atoms and no NAF-literals anymore. Thus, (L, sP^I_L) is a positive dl-program, and by Corollary 4.3, has a least model, if it is satisfiable. We thus define the strong answer set semantics of general dl-programs by reduction to the least model semantics of positive dl-programs as follows.

Definition 4.11. Let $KB = (L, P)$ be a dl-program. A *strong answer set* of KB is an interpretation $I \subseteq HB_P$ such that I is the least model of (L, sP^I_L). We denote by $ans_s(KB)$ the set of all strong answer sets of KB. If a ground literal l is in every (resp., some) strong answer set of KB, then we say that l is a *cautious* (resp., *brave*) *consequence* of KB (*under the strong answer set semantics*), in symbols $KB \models_{s,c} l$ (resp., $KB \models_{s,b} l$).

The following result shows that the strong answer set semantics of a dl-program $KB = (L, P)$ without dl-atoms coincides with the ordinary answer set semantics of P.

Theorem 4.12. *Let $KB = (L, P)$ be a dl-program without dl-atoms. Then, $I \subseteq HB_P$ is a strong answer set of KB iff it is an answer set of the ordinary program P.*

The next result shows that, as desired, strong answer sets of a dl-program KB are also models of KB, and moreover minimal models of KB if all dl-atoms are monotonic (and known as such).

Theorem 4.13. *Let $KB = (L, P)$ be a dl-program, and let M be a strong answer set of KB. Then, (a) M is a model of KB, and (b) M is a minimal model of KB if $DL_P = DL^+_P$.*

The following result shows that a positive (resp., stratified) dl-program KB is satisfiable (resp., consistent) iff it has a strong answer set. In this case, it has at most one strong answer set, which coincides with its canonical minimal model M_{KB}.

Theorem 4.14. *Let $KB = (L, P)$ be a positive (resp., stratified) dl-program. If KB is satisfiable (resp., consistent), then M_{KB} is the only strong answer set of KB. If KB is unsatisfiable (resp., inconsistent), then KB has no strong answer set.*

Since the strong answer sets of a stratified dl-program KB are independent of the stratification μ of KB, we thus obtain that the notion of consistency of KB and the canonical minimal model M_{KB} are both independent of μ.

Corollary 4.15. *Let KB be a stratified dl-program. Then, the notion of consistency of KB and the model M_{KB} do not depend on the stratification of KB.*

Example 4.16. Consider now the full dl-program of Example 4.1. This program is not stratified, in view of Rules (11) and (12), which take care of the selection between the different candidates for being reviewers. Each strong answer set containing no error flags corresponds to an acceptable review assignment scenario.

4.2.6. Weak answer set semantics of dl-programs

We finally introduce the *weak answer set semantics* of general dl-programs, which associates with a dl-program a larger set of models than the strong answer set semantics. It is based on a generalized transformation that removes all NAF-literals and *all* dl-atoms, and it reduces to the answer set semantics of ordinary programs.

Definition 4.17. Let $KB = (L, P)$ be a dl-program. The *weak dl-transform* of P relative to L and to an interpretation $I \subseteq HB_P$, denoted wP_L^I, is the ordinary positive program obtained from $ground(P)$ by deleting

(i) all dl-rules r such that either $I \not\models_L a$ for some dl-atom $a \in B^+(r)$, or $I \models_L l$ for some $l \in B^-(r)$; and
(ii) from every remaining dl-rule r all the dl-atoms in $B^+(r)$ and all the literals in $B^-(r)$.

Observe that wP_L^I is an ordinary ground positive program, which does not contain any dl-atoms anymore, and which also does not contain any NAF-literals anymore. We thus define the weak answer set semantics of general dl-programs by reduction to the least model semantics of ordinary positive programs as follows.

Definition 4.18. Let $KB = (L, P)$ be a dl-program. A *weak answer set* of KB is an interpretation $I \subseteq HB_P$ such that I is the least model of the ordinary positive program wP_L^I. We denote by $ans_w(KB)$ the set of all weak answer sets of KB. If a ground literal l is in every (resp., some) weak answer set of KB, then we say that l is a *cautious* (resp., *brave*) *consequence* of KB (*under the weak answer set semantics*), in symbols $KB \models_{s,c} l$ (resp., $KB \models_{s,b} l$).

The following result shows that the weak answer set semantics of a dl-program $KB = (L, P)$ without dl-atoms coincides with the ordinary answer set semantics of P.

Theorem 4.19. *Let $KB = (L, P)$ be a dl-program without dl-atoms. Then, $I \subseteq HB_P$ is a weak answer set of KB iff it is an answer set of the ordinary normal program P.*

The next result shows that every weak answer set of a dl-program KB is also a model of KB. In contrast to strong answer sets, a weak answer set of KB is not necessarily a minimal model, even if KB has only monotonic dl-atoms.

Theorem 4.20. *Let $KB = (L, P)$ be a dl-program. Then, every weak answer set of KB is also a model of KB.*

The following definition introduces the gl*-reduct, which transforms a given dl-program relative to an interpretation into an ordinary normal program.

Definition 4.21. Let $KB = (L, P)$ be a dl-program and $I \subseteq HB_P$. Then, the *gl*-reduct* of P relative to L and I, denoted P_L^I, is obtained from $ground(P)$ by (i) deleting every dl-rule r where either $I \not\models_L a$ for some dl-atom $a \in B^+(r)$, or $I \models_L a$ for some dl-atom $a \in B^-(r)$, and (ii) deleting from every remaining dl-rule r every dl-atom in $B^+(r) \cup B^-(r)$.

The following theorem shows that the weak answer set semantics of dl-programs can be reduced to the answer set semantics of ordinary normal programs.

Theorem 4.22. *Let $KB = (L, P)$ be a dl-program and $I \subseteq HB_P$. Then, I is a weak answer set of KB iff I is an answer set of the gl*-reduct P_L^I.*

The next theorem shows that the set of all strong answer sets of a dl-program KB is contained in the set of all weak answer sets of KB. Intuitively, the additional information about the monotonicity of dl-atoms that we use for specifying

strong answer sets allows for focusing on a smaller set of models. Hence, the set of all weak answer sets of *KB* can be seen as an approximation of the set of all strong answer sets of *KB*. Note that the converse of the following theorem generally does not hold. That is, there exist dl-programs *KB* that have a weak answer set that is not a strong answer set.

Theorem 4.23. *Every strong answer set of a dl-program $KB = (L, P)$ is also a weak answer set of KB.*

Example 4.24. Consider $P = \{p(a) \leftarrow DL[c \uplus p; c](a)\}$ and $L = \emptyset$. The unique strong answer set of (L, P) is $M_1 = \emptyset$, while the weak answer sets of (L, P) are M_1 and $M_2 = \{p(a)\}$.

It is important to observe that the weak answer set semantics does not enjoy the property of minimality of answer sets as the strong answer set semantics does (in case all dl-atoms are known to be monotonic, cf. Theorem 4.13). Thus, in the above example, M_2, although not minimal, is a weak answer set. M_2 might be considered a counterintuitive answer, since evidence of the truth of $p(a)$ is inferred by means of a "self-supporting" loop. This problem is solved by means of the strong answer set semantics. Nonetheless, when no knowledge about monotonicity of dl-atoms is available, the weak answer set semantics remains a reasonable choice.

The above problem is strictly related to the issue of establishing an intuitive semantics for logic programs with aggregates. A further discussion of this issue, focused on logic programs with aggregates, and proposing a new notion of reduct for answer set programs, is given by Faber, Leone, and Pfeifer [34].

4.3. Further semantic properties of dl-programs

We now discuss some further semantic aspects of dl-programs under the answer set semantics. We first describe in which way they are a conservative extension of their constituents. We then discuss how one can deal with equality in dl-programs. We finally concentrate on the aspect of correctly focusing dl-queries.

4.3.1. Conservativeness

We now show that dl-programs under the answer set semantics are a conservative extension of both DL knowledge bases under their first-order semantics and ordinary normal programs under the answer set semantics.

The following proposition shows that dl-queries allow for correctly querying DL knowledge bases under their first-order semantics. This result follows immediately from the semantics of dl-atoms.

Proposition 4.25. *Let $KB = (L, P)$ be a dl-program with $P = \{q(\mathbf{t}) \leftarrow DL[Q](\mathbf{t})\}$, where Q is a dl-query and q is a predicate symbol of matching arity. Furthermore, let $\mathbf{c}$ be a ground instance of $\mathbf{t}$. Then, $KB \models_{\kappa,\mu} q(\mathbf{c})$ iff $L \models Q(\mathbf{c})$, for all $\kappa \in \{w, s\}$ and $\mu \in \{c, b\}$.*

The next proposition shows that both the strong and the weak answer set semantics of a dl-program $KB = (L, P)$ without dl-atoms coincide with the ordinary answer set semantics of P—it summarizes Theorems 4.12 and 4.19.

Theorem 4.26. *Let $KB = (L, P)$ be a dl-program without dl-atoms and let l be a ground literal. Then, $ans(P) = ans_w(KB) = ans_s(KB)$. Moreover, $KB \models_{\kappa,\mu} l$ iff $P \models_{\mu} l$, for all $\kappa \in \{w, s\}$ and $\mu \in \{b, c\}$.*

4.3.2. Equality reasoning

Knowledge bases in $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ allow for equality reasoning as a first-class citizen. That is, no assumption is made about the identity of individual names, following the traditional approach of first-order logic. Logic programming, including answer set programming, was foundationally based on Herbrand's Theorem. Therefore, distinct individual names are always interpreted differently. Thus, (in)equality atoms $t_1 = t_2$, $t_1 \neq t_2$ appearing in the rules component P of a dl-program $KB = (L, P)$ can be regarded as "syntactic" (in)equality statements, while equality axioms, possibly appearing in the ontology component L of *KB*, are treated according to the description logic semantics.

However, by standard techniques, we can emulate equality in P using a *congruence* relation (see Fitting [37], Chapter 9, and Section 6.3). Furthermore, P has access to (in)equality information in L, possibly enriched as described in λ, via dl-atoms of the form $DL[\lambda; =](X, Y)$ and $DL[\lambda; \neq](X, Y)$, and one can also arbitrarily increase (in)equality knowledge in L by including '$= \uplus s$' resp. '$= \cup s$' in updates λ to L, where s is a binary predicate. This enables a variety of choices on how to match the two different equality semantics. For instance, we apply this method in Section 6.3 to emulate equality reasoning in an environment that mimics DL-safe rules [80].

4.3.3. Focusing dl-queries

Informally, dl-queries provide the rules component P of a dl-program $KB = (L, P)$ with "windows" to the ontology component L. It is now important to point out that these "windows" have a certain form and that they are independent from each other. Thus, when formulating dl-queries, one has to take some care that they are appropriately focused. We now give two examples illustrating this aspect.

Description logics allow for dealing with the existence of individuals whose identity is unknown. This is typical when existential role assertions are used, such as an axiom $(\exists R.\top)(a)$ in L. Every interpretation $\mathcal{I}$ of L contains then some individual $y \in \Delta^{\mathcal{I}}$ such that $(a^{\mathcal{I}}, y) \in R^{\mathcal{I}}$. However, the identity of y may be unknown, y may not belong to any specific concept in $\mathcal{C}$, or y may even belong to different concepts in different interpretations. Thus, in the above case, the naive dl-atom $DL[R](a, T)$ generally does not have any ground instance entailed by L, nor should it. However, the correctly focused dl-atom $DL[\exists R.\top](a)$ turns out to be naturally entailed by L.

Similarly, given $L = \{man \sqcup woman \equiv person, person(lee)\}$ and P consisting of the two naive dl-rules $p(X) \leftarrow DL[man](X)$ and $p(X) \leftarrow DL[woman](X)$, we cannot conclude $p(lee)$ from $KB = (L, P)$, since neither $man(lee)$ nor $woman(lee)$ is a consequence of L. However, if we replace the two dl-rules in $KB = (L, P)$ by the dl-rule $p(X) \leftarrow DL[man \sqcup woman](X)$, which contains a correctly focused dl-atom, then we can also naturally conclude $p(lee)$ from $KB = (L, P)$.

5. Computation

In this section, we give a fixpoint characterization for the strong answer set of satisfiable positive (resp., consistent stratified) dl-programs KB, and we show how to compute it by a finite fixpoint iteration (resp., by a sequence of finite fixpoint iterations along a stratification of KB). We also provide a general guess-and-check algorithm for computing the set of all weak answer sets of general dl-programs KB (which, by Theorem 4.23, includes the set of all strong answer sets of KB).

5.1. Fixpoint semantics

The answer set of an ordinary positive resp. stratified normal program P has a well-known fixpoint characterization in terms of an immediate consequence operator T_P, which gracefully generalizes to positive resp. stratified dl-programs. This can be exploited for a bottom-up computation of the strong answer set of such dl-programs.

5.1.1. Positive dl-programs

We first define the immediate consequence operator for dl-programs. For any (not necessarily satisfiable) dl-program $KB = (L, P)$, we define the operator T_{KB} on the subsets of HB_P as follows. For every $I \subseteq HB_P$, let

$$T_{KB}(I) = \begin{cases} HB_P, & \text{if } I \text{ is not consistent,} \\ \{H(r) \mid r \in ground(P), I \models_L l \text{ for all } l \in B(r)\}, & \text{otherwise.} \end{cases}$$

The following lemma shows that for a positive dl-program KB, the operator T_{KB} is monotonic, that is, $I \subseteq I' \subseteq HB_P$ implies $T_{KB}(I) \subseteq T_{KB}(I')$. This result is immediate from the fact that for positive dl-programs $KB = (L, P)$, every dl-atom that occurs in $ground(P)$ is monotonic relative to KB.

Lemma 5.1. *Let $KB = (L, P)$ be a positive dl-program. Then, T_{KB} is monotonic.*

The next result gives a characterization of the pre-fixpoints of T_{KB}. If KB is satisfiable, then every pre-fixpoint of T_{KB} is either a model of KB, or equal to HB_P. If KB is unsatisfiable, then HB_P is the only pre-fixpoint of T_{KB}. We recall here that $I \subseteq HB_P$ is a *pre-fixpoint* of T_{KB} iff $T_{KB}(I) \subseteq I$.

Proposition 5.2. *Let $KB = (L, P)$ be a positive dl-program. Then, $I \subseteq HB_P$ is a pre-fixpoint of T_{KB} iff I is either (a) a model of KB or (b) equal to HB_P.*

Since every monotonic operator has a least fixpoint, which coincides with its least pre-fixpoint, we immediately obtain the following corollary: The least fixpoint of T_{KB}, denoted $lfp(T_{KB})$, is given by the least model of KB, if KB is satisfiable, and by HB_P, if KB is unsatisfiable.

Corollary 5.3. *Let $KB = (L, P)$ be a positive dl-program. Then, (a) $lfp(T_{KB}) = M_{KB}$, if KB is satisfiable, and (b) $lfp(T_{KB}) = HB_P$, if KB is unsatisfiable.*

The next result shows that the least fixpoint of T_{KB} can be computed by a finite fixpoint iteration (which is based on the assumption that P and the number of constant symbols in Φ are finite). Note that for every $I \subseteq HB_P$, we define $T^i_{KB}(I) = I$, if $i = 0$, and $T^i_{KB}(I) = T_{KB}(T^{i-1}_{KB}(I))$, if $i > 0$.

Theorem 5.4. *Let KB be a positive dl-program. Then, $lfp(T_{KB}) = \bigcup_{i=1}^{n} T^i_{KB}(\emptyset) = T^n_{KB}(\emptyset)$ for some $n \geqslant 0$.*

Example 5.5. Suppose that P in $KB = (L, P)$ consists of the rules $r_1: b \leftarrow DL[S \uplus p; C](a)$ and $r_2: p(a) \leftarrow$, and L is the axiom $S \sqsubseteq C$. Then, KB is positive, and $lfp(T_{KB}) = \{p(a), b\}$, where $T^0_{KB}(\emptyset) = \emptyset$, $T^1_{KB}(\emptyset) = \{p(a)\}$, and $T^2_{KB}(\emptyset) = \{p(a), b\}$.

5.1.2. Stratified dl-programs

Using Theorem 5.4, we can characterize the answer set M_{KB} of a stratified dl-program KB by a sequence of fixpoint iterations along a stratification as follows. Let $\widehat{T}^i_{KB}(I) = T^i_{KB}(I) \cup I$, for all $i \geqslant 0$.

Theorem 5.6. *Suppose $KB = (L, P)$ has a stratification μ of length $k \geqslant 0$. Define the literal sets $M_i \subseteq HB_P$, $i \in \{-1, 0, \ldots, k\}$, as follows: $M_{-1} = \emptyset$, and*

$$M_i = \widehat{T}^{n_i}_{KB_i}(M_{i-1}), \quad \text{where } n_i \geqslant 0 \text{ such that } \widehat{T}^{n_i}_{KB_i}(M_{i-1}) = \widehat{T}^{n_i+1}_{KB_i}(M_{i-1}), i \geqslant 0.$$

Then, KB is consistent iff $M_k \neq HB_P$, and, in this case, $M_k = M_{KB}$.

Notice that $M_0 = lfp(T_{KB_0})$ and that $M_{i-1} = \widehat{T}^j_{KB_i}(M_{i-1}) \cap HB^{\star}_{P_{i-1}}$ holds for any j if $\widehat{T}^j_{KB_i}(M_{i-1})$ is consistent, which means that n_i always exists.

Example 5.7. Assume that in program P of Example 5.5 also $r_3\colon q(x) \leftarrow not\ \neg b,\ not\ DL[S](x)$ is included. Then, the mapping μ that assigns 1 to $q(a)$, 0 to $DL[S](a)$, and 0 to all other ground atoms and ground dl-atoms in $HB_P \cup DL_P$ stratifies KB, and $M_0 = lfp(T_{KB_0}) = \{p(a), b\}$ and $M_1 = \{p(a), b, q(a)\} = M_{KB}$.

5.2. General algorithm for computing weak answer sets

Computing the set of all weak answer sets of a given (general) dl-program $KB = (L, P)$ can be reduced to computing the set of all answer sets of a normal logic program. This is done by a guess-and-check algorithm as follows:

(1) We first replace each dl-atom $a(\mathbf{t})$ in P of the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t})$$

by a fresh atom $d_a(\mathbf{t})$.

(2) We then add to the result of Step (1) all ground rules of the form

$$d_a(\mathbf{c}) \leftarrow not\ \neg d_a(\mathbf{c}) \quad \text{and} \quad \neg d_a(\mathbf{c}) \leftarrow not\ d_a(\mathbf{c}) \tag{15}$$

for each dl-atom $a(\mathbf{c}) \in DL_P$. Intuitively, they "guess" the truth values of the dl-atoms of P.[3] We denote the resulting normal logic program by P_{guess}.

(3) We construct the answer sets of P_{guess} and check whether the original "guess" of the truth values of the auxiliary atoms $d_a(\mathbf{c})$ is correct relative to the given DL knowledge base L. That is, for each answer set I of P_{guess} and each dl-atom $a(\mathbf{c}) \in DL_P$, we check whether $d_a(\mathbf{c}) \in I$ iff $I \models_L a(\mathbf{c})$. If this condition holds, then $I \cap HB_P$ (which is the restriction of I to HB_P) is a weak answer set of KB.

The following theorem shows the correctness of the above algorithm.

Theorem 5.8. *Let $KB = (L, P)$ be a dl-program, and let $I \subseteq HB_P$. Then, I is a weak answer set of KB iff I can be completed to an answer set $I^{\star} \subseteq HB_{P_{\text{guess}}}$ of P_{guess} such that $d_a(\mathbf{c}) \in I^{\star}$ iff $I^{\star} \models_L a(\mathbf{c})$, for all $a(\mathbf{c}) \in DL_P$.*

Although this basic algorithm is in general not very efficient and leaves room for improvements, it shows that the weak answer set semantics can be realized on top of existing answer set solvers like DLV [23] or Smodels [81].

By Theorem 4.23, the computation of strong answer sets can be obtained by adapting the above algorithm: given a candidate weak answer set I, we generate the positive dl-program (L, sP^I_L), compute the least model M of (L, sP^I_L) by fixpoint iteration as described above, and verify that M coincides with I.

Example 5.9. The following program P (naively) emulates the closed-world assumption (see Section 6.1) on the concept *man* in the description logic base $L = \{man \sqsubseteq person, person(lee)\}$:

$$nman(X) \leftarrow not\ pman(X);$$

$$pman(X) \leftarrow DL[man \uplus nman; man](X).$$

According to the translation above, P is rewritten as:

[3] The guessing rules in (15) can be equivalently replaced by a disjunctive rule $d_a(\mathbf{c}) \vee \neg d_a(\mathbf{c}) \leftarrow$. Such disjunctive rules can be efficiently processed by the DLV system [65].

$$nman(X) \leftarrow not\ pman(X);$$

$$pman(X) \leftarrow d_1(X);$$

$$d_1(lee) \leftarrow not\ \neg d_1(lee);$$

$$\neg d_1(lee) \leftarrow not\ d_1(lee).$$

Having the two answer sets $M_1 = \{\neg d_1(lee), nman(lee)\}$ and $M_2 = \{d_1(lee), pman(lee)\}$. Note that $M_1 \not\models DL[man \uplus nman; man](lee)$ according to the fact that $\neg d_1(lee) \in M_1$, while $M_2 \not\models DL[man \uplus nman; man](lee)$, in disagreement with the fact that $d_1(lee) \in M_2$. The only accepted answer set is M_1.

The algorithms presented in this section were implemented in our system prototype, which is described in more detail in Section 8. Further details on algorithms and optimization techniques in our system prototype can be found in [95].

6. Reasoning applications

In this section, we show the usefulness of dl-programs for three concrete scenarios, where in particular the nonmonotonic behavior of the rules part is exploited in order to implement very well-known forms of nonmonotonic reasoning on top of a DL knowledge base , or as in one case to emulate another well-known extension of description logics with rules. More concretely, we show that classical forms of closed-world reasoning, like Reiter's closed-world assumption (CWA) [88] and the extended closed-world assumption (ECWA) [40,41], and of default reasoning, including Poole's [86] and Reiter-style default logic [89], can be implemented on top of a DL knowledge base . Indeed, dl-programs are particularly well-suited for emulating Reiter's default logic, since they offer the possibility to talk about consistency and provability within the language, which is a basic ingredient of this logic. Furthermore, we show that DL-safe rules [80] can be emulated in a faithful way using dl-programs.

In the rest of the section, for any DL knowledge base L, we denote by T_L its corresponding first-order theory, which we also identify with L if no confusion arises.

6.1. Closed-world reasoning

Reiter's well-known closed-world assumption (CWA) [88][4] is acknowledged as an important reasoning principle for inferring negative information from a first-order theory T: For a ground atom $p(\mathbf{c})$, conclude $\neg p(\mathbf{c})$ if $T \not\models p(\mathbf{c})$. Any such atom $p(\mathbf{c})$ is also called *free for negation*. The CWA of T, denoted $CWA(T)$, is then the extension of T with all literals $\neg p(\mathbf{c})$ where $p(\mathbf{c})$ is free for negation.

Using dl-programs, the CWA may be intuitively expressed on top of an external DL knowledge base , which can be queried through suitable dl-atoms.

Example 6.1. Consider the DL knowledge base

$$L = \{man \sqsubseteq person, person(lee)\}.$$

The ground atoms of *man* that are free for negation in L (i.e., in T_L) are determined by the following rule, where $\overline{man}$ is a fresh predicate uniquely associated with *man*:

$$\overline{man}(X) \leftarrow not\ DL[man](X). \tag{16}$$

In this case, the CWA infers $\overline{man}(lee)$.

One can set up similar rules for all the concepts and roles that are desired to be closed under CWA. Answering a query $Q(\mathbf{t})$ on L under the CWA, where Q is a (possibly negated) concept or role, is then accomplished with the stratified dl-program $KB = (L, P)$ where P contains for all concepts and roles occurring in L and Q a rule similar to (16), and a rule

$$q(\mathbf{t}) \leftarrow DL[\lambda; Q](\mathbf{t}), \tag{17}$$

where λ contains for each concept C (resp., role R) the expression $C \uplus \bar{c}$ (resp., $R \uplus \bar{r}$). The ground instances $Q(\mathbf{c})$ of $Q(\mathbf{t})$ such that $CWA(L) \models Q(\mathbf{c})$ are then given by the atoms $q(\mathbf{c})$ in the single answer set of KB. In the above example, the query $man(X)$ has no answer, while $\neg man(X)$ has the answer $X = lee$.

DL knowledge bases lack this notion of inference, adhering to the *open-world assumption*. The open-world assumption can indeed be considered reasonable in a variety of contexts, such as the Semantic Web scenario. There, a single knowledge base is generally considered as part of a distributed pool of information, rather than an isolated traditional database containing complete information. Thus, new information may easily contradict information inferred by CWA and lead to inconsistency.

[4] Throughout this section, we refer to [14,74] for references to closed-world reasoning and circumscription.

Nonetheless, it is acknowledged that many Semantic Web application scenarios require some form of closed-world reasoning [2,46,85]. Furthermore, the use of description logics for more expressive data models than the plain relational model, which has been proposed, e.g., in [8,16] and advocated for enterprise application integration and data integration [64], also increases the interest in dealing with the CWA in this context.

However, the CWA has well-known problems with inconsistency, stemming from disjunctive knowledge as noted, e.g., in [74]. Several refinements of the CWA have been proposed to avoid such inconsistency, by restricting the predicates that can safely be negated, and/or by considering more general formulas than literals to be free for negation (see, e.g., [14,74] for an overview).

One of the most advanced refinements is the *extended closed-world assumption* (ECWA) [40,41], which is intimately related to circumscription [67]. As in [40,41], in the rest of this section, we adopt the following restrictions on the theory T, which are common in a database context:

- the *domain-closure assumption* (DCA): $\forall x(x = c_1 \vee x = c_2 \vee \cdots \vee x = c_n)$, which states that there are no individuals other than the individuals $c_1, \ldots, c_n$ that are explicitly named in the theory T, and
- the *unique-names assumption* (UNA): $c_i \neq c_j$, for all $i \neq j$, which states that distinct names also refer to distinct objects in the domain.

Note that, while DCA and UNA are not necessarily true for L, the semantics given to a dl-program $KB = (L, P)$ (no matter whether L fulfills DCA and UNA or not) implicitly fulfills these assumptions. However, DCA and UNA can also be expressed in a description logics having the *oneOf* construct and the possibility to express disequality axioms, such as $\mathcal{SHOIN}(\mathbf{D})$.

The ECWA introduces a partitioning $\langle \mathcal{P}, \mathcal{Q}, \mathcal{Z} \rangle$ of the predicates in T into three disjoint lists (viewed as sets) $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{Z}$ ($\mathcal{Q}$ is often omitted, since it is clear from $\mathcal{P}$ and $\mathcal{Z}$). Informally, the predicates in $\mathcal{P}$ should be minimized, and $\neg p(\mathbf{c})$ concluded if $p(\mathbf{c})$ cannot be proved, while the predicates in $\mathcal{Q}$ are not subject to such inference, and the predicates in $\mathcal{Z}$ can take arbitrary extensions in order to minimize those in $\mathcal{P}$.

Semantically, the ECWA is characterized in terms of minimal models defined as follows. Given two interpretations M and N for T, we write $M \leqslant_{\mathcal{P},\mathcal{Z}} N$ if M and N only differ in how they interpret predicate symbols in $\mathcal{P}$ and $\mathcal{Z}$, and for each $p \in \mathcal{P}$ the extension in M is a subset of the extension in N. We call M a $\langle \mathcal{P}, \mathcal{Z} \rangle$-*minimal model* of T, iff M is a model of T and there is no model N of T such that $N \leqslant_{\mathcal{P},\mathcal{Z}} M$ and $M \not\leqslant_{\mathcal{P},\mathcal{Z}} N$. We say that $T \models_{ECWA} \phi$, if the formula ϕ is true in all $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal models of T.

Informally, a model M of T is $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal, if it makes a smallest set of ground atoms over $\mathcal{P}$ true, while the interpretation of $\mathcal{Q}$ is fixed and the atoms over $\mathcal{Z}$ may take arbitrary value. In particular, if $\mathcal{P}$ contains all predicates, then the set of $\langle \mathcal{P}, \emptyset \rangle$-*minimal models* of T corresponds to the minimal Herbrand models of T.

Since we can view a DL knowledge base L as its corresponding first-order theory T_L with unary and binary predicates for concepts and roles, respectively, we can readily apply the ECWA to it. We next describe an encoding of the $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal models for an arbitrary L.

Definition 6.2. Let L be a DL knowledge base , and let $\langle \mathcal{P}, \mathcal{Q}, \mathcal{Z} \rangle$ be a partitioning of all concepts and roles occurring in it. The dl-program $KB^{\mathbf{ECWA}}_{\langle \mathcal{P},\mathcal{Z} \rangle} = (L, P)$ is built by constructing P as follows:

(1) For each concept or role p in $\mathcal{P}$, add the rules

$$\overline{p}(\vec{X}) \leftarrow \mathit{not}\ p^+(\vec{X}), \tag{18}$$

$$p^+(\vec{X}) \leftarrow DL[\lambda; p](\vec{X}), \tag{19}$$

where λ contains for each p in $\mathcal{P} \cup \mathcal{Q}$ the expression $p \uplus \overline{p}$, and for each p in $\mathcal{Q}$ the expression $p \uplus p^+$.

(2) For each concept or role p in $\mathcal{Q} \cup \mathcal{Z}$, add the rules

$$\overline{p}(\vec{X}) \leftarrow \mathit{not}\ p^+(\vec{X}), \tag{20}$$

$$p^+(\vec{X}) \leftarrow \mathit{not}\ \overline{p}(\vec{X}). \tag{21}$$

(3) P contains the rule

$$\mathit{fail} \leftarrow DL[\lambda'; \bot](b), \mathit{not}\ \mathit{fail}, \tag{22}$$

where λ' is λ from above plus the expressions $z \uplus z^+$ and $z \uplus \bar{z}$ for each z in $\mathcal{Z}$, b is an arbitrary constant symbol, and *fail* is a fresh atom.

In this program, Rules (18) and (19) determine the extensions of the predicates from $\mathcal{P}$ in a $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal model. Here, the assumptions on $\mathcal{P}$ and $\mathcal{Q}$ are fed into L in (19), but not those on $\mathcal{Z}$, since they are not relevant. Rules (20) and (21) simply guess the extension of the concepts and roles in $\mathcal{Q}$ and $\mathcal{Z}$. The compatibility of the interpretation of all ground atoms is then checked with Rule (22); note that, by the minimality of the part on $\mathcal{P}$, no positive assumptions about $\mathcal{P}$ have to be fed into L (i.e., $p \uplus p^+$ is not needed in λ').

Theorem 6.3. *Let L be a DL knowledge base , and let $\langle \mathcal{P}, \mathcal{Q}, \mathcal{Z} \rangle$ be a partitioning of its concepts and roles. Then:*

(1) *For each strong answer set M of $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$, there exists a $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal model M' of L such that for each ground atom $p(\mathbf{c})$, $M' \models p(\mathbf{c})$ iff $p^+(\mathbf{c}) \in M$.*
(2) *For each $\langle \mathcal{P}, Z \rangle$-minimal model M' of L, there exists a strong answer set M of $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$ such that for each ground atom $p(\mathbf{c})$, $M' \models p(\mathbf{c})$ iff $p^+(\mathbf{c}) \in M$.*

Example 6.4. If we minimize in the DL knowledge base

$$L = \{man \sqsubseteq person,\ \ person \equiv man \sqcup woman,\ \ person(lee)\}$$

all predicates ($P = man, woman, person$), then we get the minimal models $M_1 = \{person(lee), man(lee)\}$ and $M_2 = \{person(lee), woman(lee)\}$. We can elegantly single out these minimal models by the strong answer sets of the following dl-program $KB' = (L, P')$:

$$\begin{aligned}
\overline{man}(X) &\leftarrow not\ man^+(X),\\
\overline{woman}(X) &\leftarrow not\ woman^+(X),\\
\overline{person}(X) &\leftarrow not\ person^+(X),\\
man^+(X) &\leftarrow DL[\lambda; man](X),\\
woman^+(X) &\leftarrow DL[\lambda; woman](X),\\
person^+(X) &\leftarrow DL[\lambda; person](X),
\end{aligned}$$

where $\lambda = woman \uplus \overline{woman}, man \uplus \overline{man}, person \uplus \overline{person}$. Intuitively, $p^+(X)$ for predicate p means that $p(X)$ is *provably true* in a minimal model to be constructed, and cannot be switched to false to generate a smaller model. The first three rules state that, by default, a ground atom $p(\mathbf{c})$ is not provably true, and thus $p(\mathbf{c})$ false in the minimal model, which is represented by $\overline{man}(\mathbf{c})$ in the strong answer set. The next three rules query, for each $p(\mathbf{c})$, whether $p(\mathbf{c})$ is provably true on L under all assumptions about non-provability of atoms. If in all cases the answer complies with the assumption, then we have a minimal model of L and a strong answer set of $\mathcal{P}$. Otherwise, the assumptions encoded in the interpretation cannot be reproduced using the rules, and we have not an answer set.

In our example, the program has two strong answer sets:

$$\begin{aligned}
M_1 &= \{person^+(lee), woman^+(lee), \overline{man}(lee)\},\\
M_2 &= \{person^+(lee), man^+(lee), \overline{woman}(lee)\},
\end{aligned}$$

which correspond to the $\langle \mathcal{P}, \emptyset \rangle$-minimal models of L as desired.

An immediate consequence of Theorem 6.3 is that we can reduce query answering from a DL knowledge base L under ECWA to cautious reasoning from $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$. For a given dl-query $Q(\mathbf{t})$, where Q is a (possibly negated) role or concept p from $\mathcal{P} \cup \mathcal{Q} \cup \mathcal{Z}$, let $dl_Q(\mathbf{t}) = p^+(\mathbf{t})$, if Q is unnegated, and let $dl_Q(\mathbf{t}) = \overline{p}(\mathbf{t})$ otherwise.

Corollary 6.5. *Let L be a DL knowledge base , let $\langle \mathcal{P}, \mathcal{Q}, \mathcal{Z} \rangle$ be a partitioning of all concepts and roles occurring in it, and let $Q(\mathbf{t})$ be a query as above. Then, for every ground instance $Q(\mathbf{c})$, $L \models_{ECWA} Q(\mathbf{c})$ iff $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle} \models_{s,c} dl_Q(\mathbf{c})$, i.e., iff $dl_Q(\mathbf{c})$ is a cautious consequence of $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$ under the strong answer set semantics.*

We finally remark that without the domain-closure assumption, the dl-program $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$ does not single out the $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal models of L in general. The reason is that $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$ selects models where the set of ground facts over P which are true is minimal. But, in general, models with different domains might be compared; unnamed individuals can arbitrarily help to minimize this set of facts, thus invalidating the traditional notion of model minimality.

6.2. Default reasoning

In essence, description logics can be viewed as a fragment of classical first-order logic in disguise, and thus share many of its properties. Among them is the property of monotonicity, according to which all conclusions remain valid if the stock of knowledge increases. In particular, DL knowledge bases only support monotonic inheritance from a more general to a more specific concept. This makes expressing "default" inheritance, according to which inheritance is carried out unless it is overridden, impossible in a direct way. However, overriding a "default" value is often a natural method for defining a subclass.

Example 6.6. Consider the following simple wine ontology L, which contains some knowledge about red and white wine, Lambrusco, as well as about Veuve Cliquot and Lambrusco di Modena.

$$L = \big\{ redWine \sqsubseteq \neg whiteWine,$$
$$lambrusco \sqsubseteq sparklingWine \sqcap redWine,$$
$$sparklingWine(veuveCliquot),\ \ lambrusco(lambrusco\_di\_Modena) \big\},$$

We know that sparkling wine is usually white; however, we cannot add the axiom $sparklingWine \sqsubseteq whiteWine$ to L without destroying L's consistency, since *lambrusco* is an exception. From L alone, we cannot conclude that Veuve Cliquot is white. What is missing is the possibility to express in L that sparkling wines are white by default. This calls for an integration of description logics with default reasoning, which is a nontrivial task in general and easily leads to undecidability [6], or, more specifically, to resort to a method of nonmonotonic inheritance reasoning [60].

Our dl-programs are a convenient framework to realize different notions of defaults on L, and thus can also be exploited for implementing default reasoning on top of an existing description logic reasoner. In the above example, we may express that sparkling wines are white by default through the following two rules:

$$white(W) \leftarrow DL[sparklingWine](W), not\ \neg white(W);$$
$$\neg white(W) \leftarrow DL[whiteWine \uplus white; \neg whiteWine](W).$$

Here, we are aiming at deriving as much positive information without causing inconsistencies, i.e., *maximizing* predicate extensions instead of *minimizing* them as proposed in the previous subsection. From these rules and L, we then can conclude $white(veuveCliquot)$ and $\neg white(lambrusco\_di\_Modena)$, as desired.

As we show in what follows, Poole's approach to default reasoning [86] and Reiter's classical default logic [89] can be realized on description logics (under restrictions) using dl-programs.

6.2.1. Poole's approach

Poole's approach views default reasoning as theory formation instead of the definition of a new logic like Reiter's. He categorizes a theory into a satisfiable set $\mathcal{F}$ of closed formulas and a set H of (possibly open) formulas, called *possible hypotheses*. The formulas in $\mathcal{F}$ are treated as "facts", which must be true in any case, while any ground instance of H *can* be used if it is consistent.

We go here one step further and equip every hypothesis g with a precondition $pc(g)$, which is another (possibly) open formula that is instantiated together with g; we denote this process by "$'$". The precondition enables the instance g' of g, if the instance $pc(g)'$ of $pc(g)$ is provable from L. Similar to Poole [86], we introduce the following notions.

Definition 6.7. Given a satisfiable set $\mathcal{F}$ of closed formulas and a set H of possible hypotheses, a *scenario* is a satisfiable set $\mathcal{F} \cup D$, where D consists of ground instances g' such that $g \in H$ and $\mathcal{F} \vdash pc(g)'$. An *extension* of $\mathcal{F}$ is the set $Cn(\mathcal{F} \cup S)$ of consequences of some maximal (with respect to $\subseteq$) scenario $\mathcal{F} \cup S$.

As usual, $\vdash$ denotes the classical derivability relation and $Cn(\mathcal{F}) = \{\phi \mid \mathcal{F} \vdash \phi \text{ and } \phi \text{ is closed}\}$, for every set $\mathcal{F}$ of closed formulas.

In what follows, we assume that $pc(g) = \ell_1(\vec{X}_1) \wedge \cdots \wedge \ell_k(\vec{X}_k)$ is a conjunction of literals $\ell_i(\vec{X}_i)$ and that g is a literal $\ell_0(\vec{X}_0)$, where the predicate of each ℓ_i is a concept or a role p_i, and $\vec{X}_i$ is a tuple of variables and constants of the arity of p_i. Intuitively, g maximizes (resp., minimizes) the extension of p_i when ℓ_i is positive (resp., negative), whenever $pc(g)$ is true in a DL knowledge base L while maintaining consistency. In Example 6.6, the possible hypothesis would be $g = whiteWine(X)$ with $pc(g) = sparklingWine(X)$, which has an instance g' resulting for $X = veuveCliquot$. We encode such defaults in a dl-program as follows.

Definition 6.8. Let L be a satisfiable DL knowledge base , and let $H = \{g_i \mid 1 \leqslant i \leqslant n\}$ be a set of possible hypotheses, where $pc(g_i) = \ell_{i,1}(\vec{X}_{i,1}) \wedge \cdots \wedge \ell_{i,k_i}(\vec{X}_{i,k_i})$ is a conjunction of literals and $g_i = \ell_{i,0}(\vec{X}_{i,0})$ is a literal with predicate p_i. Then, KB^{pl} is the dl-program (L, P), where P contains for each $i = 1, \ldots, n$, the rules:

$$g_i(\vec{X}_{i,0}) \leftarrow DL[\ell_{i,1}](\vec{X}_{i,1}), \ldots, DL[\ell_{i,k_i}](\vec{X}_{i,k_i}), not\ \neg g_i(\vec{X}_{i,0}), \tag{23}$$
$$\neg g_i(\vec{X}_{i,0}) \leftarrow DL[\lambda; \neg\ell_{i,0}](\vec{X}_{i,0}), \tag{24}$$

where g_i is a predicate in the logic program for p_i, and $\lambda = p_1\, op_1\, g_1, \ldots, p_n\, op_n\, g_n$, where $op_i = \uplus$ if the literal ℓ_{i_0} is positive and $op_i = \cup$ otherwise (and double negation in $\neg\ell_{i,0}$ is canceled).

The answer set semantics effects that the update λ of L is maximal and, moreover, preserves consistency. If it would cause inconsistency, then $\neg g_i(\mathbf{c}_{i,0})$ would be derived for every instance $(g_i)'$ of g_i where $X_{i,0} = \mathbf{c}_{i,0}$, and hence no rule of

form (23) could be applied; thus, the update would be empty, and L would have to be inconsistent itself. Note that the encoding of the default in Example 6.6 is of this form.

Formally, we have the following correspondence result.

Theorem 6.9. *Let L and $H = \{g_i \mid 1 \leqslant i \leqslant n\}$ be as in Definition 6.8. For every interpretation M, let $scen(M) = L \cup \{\ell_{i,0}(\mathbf{c}_{i,0}) \mid g_i(\mathbf{c}) \in M\}$. Then:*

(1) *For every strong answer set M of KB^{pl}, $scen(M)$ is a maximal scenario.*
(2) *For every maximal scenario $L \cup S$, there exists a strong answer set M of KB^{pl} such that $L \cup S = scen(M)$.*

In our Example 6.6, we have a single strong answer set of KB^{pl}, which is $M = \{g(veuveCliquot), \neg g(lambrusco\_di\_Modena)\}$; the corresponding maximal scenario is $L \cup \{whiteWine(veuveCliquot)\}$.

By adding to the dl-program in Definition 6.8 the rules

$$p^+(\vec{X}) \leftarrow DL[\lambda; p](\vec{X}), \qquad \neg p^+(\vec{X}) \leftarrow DL[\lambda; \neg p](\vec{X}), \tag{25}$$

where p is a concept or a role, we can export positive resp. negative ground literals on p from the extension of a maximal scenario to the corresponding answer set. In the scenario of Example 6.6, the rule

$$redWine^+(W) \leftarrow DL[whiteWine \uplus white; redWine](W)$$

would export the red wines, and in particular $redWine^+(lambrusco\_di\_Modena)$ would be contained in the single answer set.

We can also exploit this for expressing brave and cautious query answering from the extensions of L. Given a dl-query $Q(\mathbf{t})$, where Q is a (possibly negated) concept or role p, in presence of the respective rule (25), for each instance $Q(\mathbf{c})$, it holds that the literal $(\neg)p(\mathbf{c})$ belongs to some (resp., every) extension of L iff $(\neg)p^+(\mathbf{c})$ belongs to some (resp., every) strong answer set of KB.

It is also possible to encode more general defaults than literals into dl-programs, using a different technique. If $g = \mu_1(\vec{X}_1) \wedge \cdots \wedge \mu_n(\vec{X}_n)$ is a conjunction of literals $\mu_i(\vec{X}_i)$ with predicates p_i, we can emulate for each instance g' of g the test whether $L \cup S \cup \{\mu_1(\mathbf{c}_1) \wedge \cdots \wedge \mu_n(\mathbf{c}_n)\}$ is satisfiable for a scenario $L \cup S$ (represented by an update λ), by an extended update $\lambda_{g'}$ using a dl-literal *not* $DL[\lambda_{g'}; \bot](b)$ (see Appendix D). For open defaults, this may require many rules (in general, exponentially many in the number of variables) and thus will not be very efficient. Using a similar technique, also disjunctive preconditions can be expressed, and more generally preconditions in conjunctive normal form. However, encoding disjunction in the possible hypothesis itself for a correspondence similar as in Theorem 6.9 seems infeasible (note that dl-programs lack the possibility to add disjunctions to L).

While the precondition $pc(g)$ allows us to selectively apply a default g (like in case of the white-wine default), Definition 6.7 does not capture "chaining" of defaults, where one default has to be applied to enable the application of another one. In this way, expected conclusions might be missed.

Example 6.10. Suppose that in the wine scenario, we further know that white wine is usually served cold, expressed by the default $h = servedCold(X)$ with $pc(h) = whiteWine(X)$. From the program KB^{pl} in Theorem 6.9, however, we cannot conclude that Veuve Cliquot is served cold, since $h(veuveCliquot)$ is not in the single strong answer set of KB^{pl}. The reason is that the precondition of h for the instance $X = veuveCliquot$, $whiteWine(veuveCliquot)$, is not a consequence of L, but is only obtained after the application of the default g instantiated to $veuveCliquot$.

In order to propagate conclusions from defaults to preconditions of defaults, we have to add these conclusions to the DL knowledge base L when testing the preconditions. To this end, we add in each dl-atom $DL[\ell_{i,j}](\vec{X}_{i,j})$ in rule (23) the update λ. Following this approach, we can in fact realize a semantics of defaults as in Reiter's Default Logic [89], which we show next.

6.2.2. Reiter's default logic

Recall that a default theory $\Delta = \langle W, D\rangle$ consists of a set W of first-order sentences and a set D of defaults of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}, \tag{26}$$

(also written $\frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma}$) where α, all β_i, and γ are (possibly open) first-order formulas. Reiter defines the extensions of a closed default theory Δ (i.e., where all defaults in Δ contain sentences only) as the fixpoints of the operator $\Gamma_\Delta(S)$ as follows. For a set of sentences S, $\Gamma_\Delta(S)$ is the least set of sentences such that

(1) $W \subseteq \Gamma_\Delta(S)$;
(2) $Cn(\Gamma_\Delta(S)) = \Gamma_\Delta(S)$;
(3) if $\frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma} \in D$, $\alpha \in \Gamma_\Delta(S)$, and $\neg\beta_1, \ldots, \neg\beta_n \notin S$ then $\gamma \in \Gamma_\Delta(S)$.

Then, a set of formulas E is an extension of Δ iff $E = \Gamma_\Delta(E)$. Extensions of *open* default theories (i.e., default theories which are not closed) are defined via ground instances of defaults. In case the defaults do not contain quantifiers, the grounding is defined analogously to logic program rules; otherwise, a suitable skolemization step is required (see [74,89] for details).[5]

One can express the extensions of a default theory $\Delta = \langle L, D\rangle$, where L is a DL knowledge base, and D consists of certain quantifier-free defaults, by the strong answer sets of a corresponding dl-program. We show this here for defaults of form (26) where α is a conjunction of literals and all β_i's and γ are literals (possible generalizations are discussed at the end of this subsection).

Roughly speaking, we encode Δ to a dl-program that implements a guess-and-check strategy, by which a sufficiently large part of an extension E, including all conclusions γ of applied defaults δ, is guessed using predicates in_γ and out_γ and described by an update λ' of L, in which we have $p \uplus in\_\gamma$, where p is the predicate of γ, if the literal γ is positive and $p \cup in\_\gamma$ otherwise. The candidate E is then checked using a predicate g for γ to characterize $\Gamma_\Delta(E)$, which is described by an update λ of L which includes $p \uplus g$ if γ is a positive literal and $p \cup g$ otherwise.

Definition 6.11. Let L be a DL knowledge base, and let $D = \{\delta_1, \ldots, \delta_n\}$ be a set of quantifier-free defaults of the form $\delta_i = \frac{\alpha_i : \beta_{i,1}(\vec{Z}_{i,1}), \ldots, \beta_{i,n_i}(\vec{Z}_{i,n_i})}{\gamma_i(\vec{Y}_i)}$ where $\alpha_i = \alpha_{i,1}(\vec{X}_{i,1}) \wedge \cdots \wedge \alpha_{i,k_i}(\vec{X}_{i,k_i})$ is a conjunction of literals $\alpha_{i,j}(\vec{X}_{i,j})$, all $\beta_{i,j}(\vec{Z}_{i,j})$ are literals, and $\gamma_i(\vec{Y}_i)$ is a literal with predicate p_i. Then, KB^{df} is the dl-program (L, P), where P contains for each $i = 1, \ldots, n$, the following rules:

(1) rules that guess whether δ_i's conclusion $\gamma_i(\vec{Y}_i)$ belongs to the extension E:

$$in\_\gamma_i(\vec{Y}_i) \leftarrow not\ out\_\gamma_i(\vec{Y}_i), \tag{27}$$

$$out\_\gamma_i(\vec{Y}_i) \leftarrow not\ in\_\gamma_i(\vec{Y}_i); \tag{28}$$

(2) a rule which checks the compliance of the guess for E with L:

$$fail \leftarrow DL[\lambda'; \gamma_i](\vec{Y}_i), out\_\gamma_i(\vec{Y}_i), not\ fail, \tag{29}$$

where $\lambda' = p_1\ op_1\ in\_\gamma_1, \ldots, p_n\ op_n\ in\_\gamma_n$, and $op_j = \uplus$ if the literal $\gamma_j(\vec{Y}_j)$ is positive and $op_j = \cup$ otherwise, $1 \leqslant j \leqslant n$;

(3) a rule for applying δ_i as in $\Gamma_\Delta(E)$:

$$g_i(\vec{Y}_i) \leftarrow DL[\lambda; \alpha_{i,1}](\vec{X}_{i,1}), \ldots, DL[\lambda; \alpha_{i,k_i}](\vec{X}_{i,k_i}), \\ not\ DL[\lambda'; \neg\beta_{i,1}](\vec{Z}_{i,1}), \ldots, not\ DL[\lambda'; \neg\beta_{i,n_i}](\vec{Z}_{i,n_i}), \tag{30}$$

where $\lambda = p_1\ op_1\ g_1, \ldots, p_n\ op_n\ g_n$ and $op_j = \uplus$ if the literal $\gamma_j(\vec{Y}_j)$ is positive, and $op_j = \cup$ otherwise, $1 \leqslant j \leqslant n$ (and double negation in $\neg\beta_{i,h}$ is canceled).

(4) rules which check whether E and $\Gamma_\Delta(E)$ coincide:

$$fail \leftarrow not\ DL[\lambda; \gamma_i](\vec{Y}_i), in\_\gamma_i(\vec{Y}_i), not\ fail, \tag{31}$$

$$fail \leftarrow DL[\lambda; \gamma_i](\vec{Y}_i), out\_\gamma_i(\vec{Y}_i), not\ fail. \tag{32}$$

The following result establishes the correspondence between default extensions and strong answer sets of the program KB^{df}.[6]

Theorem 6.12. *Let $\Delta = \langle L, D\rangle$ be a default theory, where L is a DL knowledge base and $D = \{\delta_1, \ldots, \delta\}$ is a set of defaults as in Definition* 6.11. *Then*:

(1) *For each extension E of Δ, there exists a (unique) strong answer set M of KB^{df} such that*

$$E = Cn\big(L(M; \lambda')\big)\big(= Cn(L(M; \lambda))\big).$$

(2) *For each strong answer set M of KB^{df}, the set*

$$E = Cn\big(L(M; \lambda')\big)\big(= Cn(L(M; \lambda))\big)$$

is an extension of Δ.

[5] Notice that the world knowledge L is not skolemized in this paper, so open defaults are actually grounded only with respect to the constants occurring in L. This version of defaults is equivalent to the one considered by Baader and Hollunder [6].

[6] This encoding can be simplified and optimized. We use this version as it features a natural guess-and-check approach, in which the guessing part (expressed by λ') and the checking part (expressed by λ) are clearly separated.

Furthermore, for each ground instance $\gamma_i(\mathbf{c})$ of $\gamma_i(\vec{Y}_i)$, $g_i(\mathbf{c}_i)$ is in a strong answer set M of KB^{df} iff $\gamma_i(\mathbf{c}_i)$ is in the corresponding extension.

Example 6.13. In the extended wine scenario, we have the two defaults

$$\delta_1 = \frac{sparklingWine(X) : whiteWine(X)}{whiteWine(X)} \quad \text{and}$$
$$\delta_2 = \frac{whiteWine(X) : servedCold(X)}{servedCold(X)}.$$

The program KB^{df} (which we omit here for space reasons) has a single answer set M, which contains the atoms $g_1(veuveCliquot)$ and $g_2(veuveCliquot)$, but neither the atom $g_1(lambrusco\_di\_Modena)$ nor the atom $g_2(lambrusco\_di\_Modena)$.

By adding rules (25) for concepts resp. roles p to KB^{df}, we can again export positive resp. negative ground literals on p from the extension to the corresponding answer set, and we can utilize this for brave and cautious reasoning from the extensions of Δ, via brave and cautious reasoning from KB^{df}. If in our example, we add the rules for $p = servedCold$, then we have in the single answer set M the literal $servedCold^+(veuveCliquot)$, but neither $servedCold^+(lambrusco\_di\_Modena)$ nor $\neg servedCold^+(lambrusco\_di\_Modena)$. Thus, we conclude (under both brave and cautious reasoning) that Veuve Cliquot is served cold, but we conclude nothing about whether Lambrusco di Modena is served cold or not.

We note that the rules (29) can be eliminated from KB^{df}, at the price of introducing more dl-literals, by replacing in (31) $in\_\gamma_i(\vec{Y}_i)$ with $DL[\lambda'; \gamma_i](\vec{Y}_i)$ and in (32) $out\_\gamma_i(\vec{Y}_i)$ with $not\ DL[\lambda; \gamma_i](\vec{Y}_i)$.

It is well-known that extended logic programs correspond to Reiter's default logic, which can be viewed as a fragment of the latter; this was already shown by Gelfond and Lifschitz in their seminal paper [39], by identifying a rule of form $a \leftarrow b_1, \ldots, b_m, not\ b_{m+1}, \ldots, not\ b_n$ with the default $\frac{b_1 \wedge \cdots \wedge b_m : \neg\beta_{m+1}, \ldots, \neg\beta_n}{a}$. However, this correspondence does not have any background theory W, i.e., default theories are of the form $(\emptyset, D)$. The encoding of default rules on top of a DL knowledge base L from above is more general, since it allows one also to handle a nonempty background theory W $(= L)$. Note also that it is different in spirit from the encoding in [39], and intuitively has to be so, since we must take inferences in the background knowledge W, also in interaction with the defaults, into account while we are bound to the evaluation mechanism of dl-programs.

We remark that the encoding KB^{df} can be generalized, using techniques discussed in Section 6.2.1 and Appendix D, to quantifier-free defaults (26) where α is in conjunctive normal form, all β_i are in disjunctive normal form, and $\gamma = \gamma_1 \wedge \cdots \wedge \gamma_k$ is a conjunction of literals γ_i (note that any such default can be efficiently rewritten to k defaults $\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma_1}, \ldots, \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma_k}$). In particular, it can be generalized to all disjunction-free defaults. However, disjunction in the conclusion γ of (26) again causes problems (as we cannot add disjunctive formulas to L via dl-atoms), and prevents a similar correspondence as in Theorem 6.12.

6.3. DL-safe rules

After CARIN [66] and $\mathcal{AL}$-*log* [21], DL-safe rules [79,80] are a further coupling of rules and ontologies while keeping a full first-order semantics together with decidability. To ensure this, only a limited form of rules is allowed.

Intuitively, a *DL-safe program* is a DL knowledge base L coupled with a set of Horn rules P. Concepts and roles from L may freely appear in P (also in rule heads). Nonetheless, any variable must appear in the body of a rule within an atom whose predicate name does not appear in L.

Definition 6.14. Suppose L is a DL knowledge base in $\mathcal{SHOIN}(\mathbf{D})$, where $\mathbf{A}$, $\mathbf{R}_A$, and $\mathbf{R}_D$ are the atomic concepts, abstract roles, and datatype roles, respectively. Let $\mathbf{P}$ be a set of predicate symbols such that $\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D \subseteq \mathbf{P}$. A (disjunctive) DL-safe rule is a (disjunctive) rule r of the form

$$h_1(\vec{Y}_1) \vee \cdots \vee h_m(\vec{Y}_m) \leftarrow b_1(\vec{X}_1), \ldots, b_n(\vec{X}_n), \tag{33}$$

where all h_i, b_j are from $\mathbf{P}$ and all $\vec{Y}_i$, $\vec{X}_j$ are lists of variables and constants matching the arities of h_i resp. b_j,[7] such that each variable in r occurs in some atom $b_j(\vec{X}_j)$ where $b_j \in \mathbf{P} \setminus (\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D)$. A *combined knowledge base* is any pair (L, P), where L is a DL knowledge base and P is a finite set of (disjunctive) DL-safe rules.

Example 6.15. Consider the simple person knowledge base L from Section 6.1, and suppose there is also a role *parent* and an atomic concept *allDaughters*, and that L contains axioms effecting $allDaughters \equiv \exists parent.\top \sqcap \forall parent.woman$, i.e., *allDaughters* are those parents whose children are all girls. Let P contain the rule

$$p(X) \leftarrow knows(X, Y), allDaughters(Y), knows(X, Z), parent(Y, Z) \tag{34}$$

[7] Atomic concepts from $\mathbf{A}$ are unary predicates, while roles from $\mathbf{R}_A \cup \mathbf{R}_D$ are binary predicates.

plus facts of the form $knows(n, n')$, where n and n' are person names. Intuitively, the rule singles out those persons who know a parent whose children are all girls, and know at least one of these children. Then, (L, P) is a combined knowledge base.

The semantics of combined knowledge bases is defined as follows.

Definition 6.16. An arbitrary first-order interpretation $\mathcal{I}$ is a *model* of (or *satisfies*) a combined knowledge base (L, P), denoted $\mathcal{I} \models (L, P)$, if $\mathcal{I} \models L$ and $\mathcal{I} \models P$.

We can simulate combined knowledge bases using dl-programs as follows.

Definition 6.17. Given a combined knowledge base (L, P), let KB^{dls} be the dl-program (L, P^{dls}), where P^{dls} includes the following rules:

(1) for each predicate p appearing in P, the rules

$$p^+(\vec{X}) \leftarrow not\ p^-(\vec{X}), \tag{35}$$

$$p^-(\vec{X}) \leftarrow not\ p^+(\vec{X}), \tag{36}$$

where p^+ and p^- are new predicates;[8]

(2) for each rule $r : h_1(\vec{Y}_1) \vee \cdots \vee h_m(\vec{Y}_m) \leftarrow b_1(\vec{X}_1), \ldots, b_n(\vec{X}_n)$ in P, the rule

$$fail \leftarrow not\ h_1^+(\vec{Y}_1), \ldots, not\ h_m^+(\vec{Y}_m), b_1^+(\vec{X}_1), \ldots, b_n^+(\vec{X}_n), not\ fail; \tag{37}$$

(3) and the rule

$$fail \leftarrow DL[\lambda; \bot](b), not\ fail, \tag{38}$$

where b is an arbitrary constant symbol and $\lambda = p_1 \uplus p_1^+, p_1 \uplus p_1^-, \ldots, p_n \uplus p_n^+, p_n \uplus p_n^-$ where $p_1, \ldots, p_n$ are all predicates in P that occur in L.

Intuitively, rules (35) and (36) guess the extension of each predicate in P. Rule (37) checks satisfaction of the rule r, and rule (38) implements a consistency check and discards guesses that are not compliant with L.

The following lemma shows that the grounding of P over the constant symbols in the language, which we denote by $P\downarrow$,[9] is sufficient to capture the models of a combined knowledge base (L, P) with respect to ground atoms.

Lemma 6.18. *Let (L, P) be a combined knowledge base. Then, for every model $\mathcal{I}$ of $(L, P\downarrow)$, there is a model $\mathcal{J}$ of (L, P) which differs from $\mathcal{I}$ only by the interpretation of predicates $p \in \mathbf{P} \setminus (\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D)$ such that for all ground atoms α, $\mathcal{J} \models \alpha$ iff $\mathcal{I} \models \alpha$.*

Indeed, this holds since we can simply remove all tuples $\mathbf{e}$ from the extensions of all predicates $p \in \mathbf{P} \setminus (\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D)$ in $\mathcal{I}$ which contain some unnamed individual, i.e., there is some element in $\mathbf{e}$ such that $\mathcal{I}$ maps no constant to it; then, $\mathcal{J}$ clearly satisfies L, and by DL-safety, all formulas in P will be satisfied. Based on this lemma, we can establish the following property of the encoding KB^{dls}.

Theorem 6.19. *Let (L, P) be a combined knowledge base, and let $ga(P)$ be the set of ground atoms with a predicate name occurring in P. Then,*

(1) *for every strong answer set M of KB^{dls}, there exists some first-order model $\mathcal{I}$ of (L, P) such that for every $p(\mathbf{c}) \in ga(P)$, $\mathcal{I} \models p(\mathbf{c})$ iff $p^+(\mathbf{c})$, and*

(2) *for every first-order model $\mathcal{I}$ of (L, P), the set*

$$M = \{p^+(\mathbf{c}) \mid p(\mathbf{c}) \in ga(P), \mathcal{I} \models p(\mathbf{c})\} \cup \{p^-(\mathbf{c}) \mid p(\mathbf{c}) \in ga(P), I \not\models p(\mathbf{c})\}$$

is a strong answer set of KB^{dls}.

Answering a query $Q(\mathbf{t})$, where Q is a (possibly negated) predicate name from $\mathbf{P}$ and $\mathbf{t}$ a list of variables and constants (resp., values), from a combined knowledge base (L, P), i.e., determining all tuples $\mathbf{c}$ of constant symbols (resp., values) such that $(L, P) \models Q(\mathbf{c})$, can then be performed as follows. Add to KB^{dls} the rule

$$q(\mathbf{t}) \leftarrow \chi(Q; \mathbf{t}), \tag{39}$$

[8] For simplicity, we do not distinguish between individuals (constants) and datatype values, whose sort can be expressed by respective typing predicates.

[9] We assume here sorted (finite) sets of constant symbols resp. values. An extension to infinite sets would not be a problem in principle, if infinite answer sets would be considered.

where q is a fresh predicate and $\chi(Q;\mathbf{t}) = DL[\lambda; Q](\mathbf{t})$, if the predicate of Q is from $\mathbf{A} \cup \mathbf{R}_A \cup \mathbf{R}_D$ but does not occur in P; otherwise, $\chi(Q;\mathbf{t}) = p^+(\mathbf{t})$, if Q is unnegated, and $\chi(Q;\mathbf{t}) = p^-(\mathbf{t})$, if Q is negated. Denote by $KB^{dls}_{Q(\mathbf{t})}$ he resulting dl-program. From Theorem 6.19, the following result is then easily obtained.

Corollary 6.20. *Given a combined knowledge base (L, P) and a query $Q(\mathbf{t})$ as above, $\mathbf{c}$ is an answer to $Q(\mathbf{t})$ iff $q(\mathbf{c})$ is a cautious consequence of $KB^{dls}_{Q(\mathbf{t})}$, i.e., belongs to all strong answer sets of $KB^{dls}_{Q(\mathbf{t})}$.*

Note that as for query answering, the rule (38) can be dropped from $KB^{dls}_{Q(\mathbf{t})}$.

Treatment of equality. The full version of [79] explicitly considers the possibility of having a binary equality predicate "$\approx$" available [80]. Viewing $\approx$ as a congruence, i.e., as an equivalence relation that is compliant with the other relations in each model (as done e.g. also in [76]), we can emulate it as follows. Suppose that "$\approx^+$" and "$\approx^-$" are fresh binary predicate names in P.

- We add to KB^{dls} the rules

$$X \approx^+ X \leftarrow,$$
$$X \approx^+ Y \leftarrow Y \approx^+ X,$$
$$X \approx^+ Y \leftarrow X \approx^+ Y, Y \approx^+ Z,$$
$$X \approx^+ Y \leftarrow \mathit{not}\ X \approx^- Y,$$
$$X \approx^- Y \leftarrow \mathit{not}\ X \approx^+ Y.$$

  These rules will effect that $\approx^+$ is, in any answer set, an equivalence relation.[10]
- For each predicate p in $\mathbf{P}$, we add to KB^{dls} a *congruence constraint*

$$\mathit{fail} \leftarrow p(\mathbf{X}), \mathit{not}\ p(\mathbf{Y}), X_1 \approx^+ Y_1, \ldots, X_n \approx^+ Y_n, \mathit{not}\ \mathit{fail},$$

  where $\mathbf{X} = X_1 \ldots, X_n$ and $\mathbf{Y} = Y_1, \ldots . Y_n$. This enforces that equal objects behave equally.
- We add to the update description λ in rule (38) the operations $\approx \uplus \approx^+$ and $\approx \uplus \approx^-$ (assuming that $\approx$ is the equality relation $=$ in L).

Then, in the strong answer sets M of the modified KB^{dls}, the predicate $\approx^+$ coincides with the relation $\approx$ in the corresponding first-order models $\mathcal{I}$ of the description logic part on all ground atoms.

7. Complexity

In this section, we address the complexity of dl-programs. We first recall the complexity classes that we encounter. We then formally state the considered reasoning problems for dl-programs and summarize relevant previous complexity results. We finally provide our complexity results for dl-programs.

7.1. Complexity classes

We assume that the reader has some elementary background in complexity theory, and is familiar with the concepts of Turing machines and oracle calls, polynomial-time transformations among problems, and the hardness and completeness of a problem for a complexity class, as can be found, e.g., in [62,63,84]. We now briefly recall the complexity classes that we encounter in our complexity results below.

The class EXP (resp., NEXP) contains all decision problems that can be solved in exponential time on a deterministic (resp., nondeterministic) Turing machine. The class co-NEXP is the complementary class of NEXP, which has yes- and no-instances interchanged, while the class $D^{exp} = \{L \times L' \mid L \in \text{NEXP}, L' \in \text{co-NEXP}\}$ is the "conjunction" of NEXP and co-NEXP. The class P^{NEXP} contains all problems that are decidable in polynomial time on a deterministic Turing machine with the help of a NEXP oracle. It coincides with the class NP^{NEXP} [47] of all problems that are decidable in polynomial time on a nondeterministic Turing machine with the help of an oracle for NEXP. The above complexity classes and their inclusion relationships (which are all currently believed to be strict) are shown in Fig. 1.

[10] To reduce guesses for $\approx^+$ in advance to those compatible with $\approx$ in L, further rules $X \approx^+ Y \leftarrow DL[=](X, Y)$ and $X \approx^- Y \leftarrow DL[\neq](X, Y)$ might be added to import all ground literals of $\approx$ that are provable from L directly (assuming "$\approx$" is equality $=$ in L).

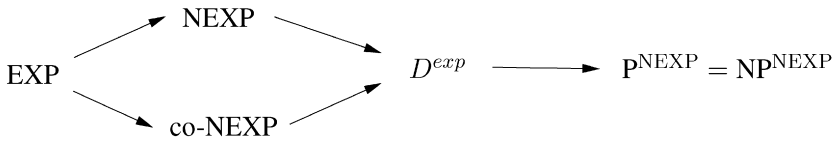

Fig. 1. Containment between complexity classes.

7.2. Problem statements and previous results

We consider the following canonical decision problems for dl-programs:

ANSWER SET EXISTENCE: Given vocabulary Φ and a dl-program $KB = (L, P)$, decide whether KB has a strong (resp., weak) answer set.

CAUTIOUS REASONING: Given vocabulary Φ, a dl-program $KB = (L, P)$, and a literal $l \in HB_P$, decide whether l is in every strong (resp., weak) answer set of KB.

BRAVE REASONING: Given vocabulary Φ, a dl-program $KB = (L, P)$, and a literal $l \in HB_P$, decide whether l is in some strong (resp., weak) answer set of KB.

We next summarize some relevant previous complexity results. We recall that deciding whether a given (non-ground) normal logic program has an answer set is complete for NEXP [17]. Furthermore, deciding whether a DL knowledge base L in $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$) is satisfiable is complete for EXP [54,98] (resp., NEXP, for both unary and binary number encoding; see [54,87] and the NEXP-hardness proof for $\mathcal{ALCQI}$ in [98], which implies the NEXP-hardness of $\mathcal{SHOIN}(\mathbf{D})$). As an easy consequence, evaluating a given ground dl-atom a of the form (1) in a given dl-program $KB = (L, P)$ and an interpretation I_p of its input predicates $p = p_1, \ldots, p_m$ (that is, deciding whether $I \models_L a$ for each I which coincides on p with I_p) is complete for EXP (resp., co-NEXP) for L from $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$).

7.3. Answer set existence

We first consider the problem of deciding whether a given dl-program $KB = (L, P)$ has a strong or weak answer set. Table 1 compactly summarizes our complexity results for this problem for L from $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. In detail, for L in $\mathcal{SHIF}(\mathbf{D})$, this problem is EXP-complete for positive and stratified KB, and NEXP-complete for general KB. For L in $\mathcal{SHOIN}(\mathbf{D})$, the problem is NEXP-complete for positive KB, and $\mathrm{P}^{\mathrm{NEXP}}$-complete for stratified and general KB. Thus, the complexity of dl-programs is not or only mildly higher than the one of its components, with the exception of general dl-programs with L from $\mathcal{SHIF}(\mathbf{D})$, where it moves from EXP to NEXP.

As for practical concerns, the complexity can be drastically lower if both components have lower complexity. For example, if evaluating dl-atoms is feasible with an NP oracle in polynomial time and the number of variables in each rule in P is bounded by a constant (e.g., if P is fixed), then deciding strong and weak answer set existence is feasible within $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$, and thus within the bounds of many classical formalisms for nonmonotonic reasoning in the propositional case [24,43]; we leave a detailed study of the complexity of fragments of dl-programs for further work. Furthermore, we stress that the results account for the worst case complexity. Even if in this case, evaluating a single ground dl-atom takes double exponential time by currently used methods, knowledge bases encountered in practice may not show this behavior and efficient DL-engines can usually evaluate dl-atoms much faster on them; this is at least our experience [29].

The following theorem shows that deciding the existence of strong or weak answer sets of dl-programs $KB = (L, P)$ with L in $\mathcal{SHIF}(\mathbf{D})$ is complete for EXP in the positive and the stratified case, and complete for NEXP in the general case.

Theorem 7.1. *Given vocabulary Φ and a dl-program $KB = (L, P)$ with L belonging to $\mathcal{SHIF}(\mathbf{D})$, deciding whether KB has a strong or weak answer set is EXP-complete when KB is positive or stratified, and NEXP-complete when KB is a general dl-program.*

The next theorem shows that deciding the existence of strong or weak answer sets of dl-programs $KB = (L, P)$ with L in $\mathcal{SHOIN}(\mathbf{D})$ is complete for NEXP in the positive case, and complete for $\mathrm{P}^{\mathrm{NEXP}}$ in the stratified and the general case.

Theorem 7.2. *Given vocabulary Φ and a dl-program $KB = (L, P)$ with L belonging to $\mathcal{SHOIN}(\mathbf{D})$, deciding whether KB has a strong or weak answer set is NEXP-complete when KB is positive, and $\mathrm{P}^{\mathrm{NEXP}}$-complete when KB is a stratified or general dl-program.*

A more detailed discussion of these and the other complexity results in this section is given in Appendix E.

7.4. Cautious and brave reasoning

We next consider the problems of cautious and brave reasoning from dl-programs, that is, of deciding whether a classical literal $l \in HB_P$ belongs to every resp. some strong or weak answer set of a given dl-program $KB = (L, P)$. Tables 2 and 3, respectively, compactly summarize our complexity results for these problems for L from $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. Roughly

Table 1
Complexity of deciding strong or weak answer set existence for dl-programs

dl-program $KB = (L, P)$	L in $\mathcal{SHIF}(\mathbf{D})$	L in $\mathcal{SHOIN}(\mathbf{D})$
KB positive	EXP-complete	NEXP-complete
KB stratified	EXP-complete	$\mathrm{P}^{\mathrm{NEXP}}$-complete
KB general	NEXP-complete	$\mathrm{P}^{\mathrm{NEXP}}$-complete

Table 2
Complexity of cautious reasoning from the strong or weak answer sets of a dl-program

$KB = (L, P)$	L in $\mathcal{SHIF}(\mathbf{D})$	L in $\mathcal{SHOIN}(\mathbf{D})$
KB positive	EXP-complete	co-NEXP-complete
KB stratified	EXP-complete	$\mathrm{P}^{\mathrm{NEXP}}$-complete
KB general	co-NEXP-complete	$\mathrm{P}^{\mathrm{NEXP}}$-complete

Table 3
Complexity of brave reasoning from the strong / weak answer sets of a dl-program

$KB = (L, P)$	L in $\mathcal{SHIF}(\mathbf{D})$	L in $\mathcal{SHOIN}(\mathbf{D})$
KB positive	EXP-complete	D^{exp}-complete/$\mathrm{P}^{\mathrm{NEXP}}$-complete
KB stratified	EXP-complete	$\mathrm{P}^{\mathrm{NEXP}}$-complete
KB general	NEXP-complete	$\mathrm{P}^{\mathrm{NEXP}}$-complete

speaking, except for brave reasoning from positive dl-programs $KB = (L, P)$ with L from $\mathcal{SHOIN}(\mathbf{D})$, the complexity of cautious (resp., brave) reasoning from dl-programs coincides with the complexity of answer set non-existence (resp., existence) for dl-programs (see Table 1).

The following theorem shows that deciding whether a classical literal $l \in HB_P$ belongs to every (resp., some) strong or weak answer set of a given dl-program $KB = (L, P)$ with L in $\mathcal{SHIF}(\mathbf{D})$ is complete for EXP in the positive and the stratified case, and complete for co-NEXP (resp., NEXP) in the general case.

Theorem 7.3. *Given vocabulary Φ, a dl-program $KB = (L, P)$ with L belonging to $\mathcal{SHIF}(\mathbf{D})$, and a classical literal $l \in HB_P$, deciding whether l is in every (resp., some) strong or weak answer set of KB is complete for EXP when KB is positive or stratified, and complete for co-NEXP (resp., NEXP) when KB is a general dl-program.*

The next theorem shows that deciding whether a classical literal $l \in HB_P$ belongs to every (resp., some) strong / weak answer set of a given dl-program $KB = (L, P)$ with L in $\mathcal{SHOIN}(\mathbf{D})$ is complete for co-NEXP (resp., $D^{exp}/\mathrm{P}^{\mathrm{NEXP}}$) in the positive case, and complete for $\mathrm{P}^{\mathrm{NEXP}}$ in the stratified and the general case. Note that brave reasoning from the weak answer sets of a positive dl-program $KB = (L, P)$ with L in $\mathcal{SHOIN}(\mathbf{D})$ is co-NEXP-complete when P is "¬"-free.

Theorem 7.4. *Given vocabulary Φ, a dl-program $KB = (L, P)$ with L belonging to $\mathcal{SHOIN}(\mathbf{D})$, and a classical literal $l \in HB_P$, deciding whether l is in every (resp., some) strong or weak answer set of KB is complete for co-NEXP (resp., $D^{exp}/\mathrm{P}^{\mathrm{NEXP}}$) when KB is positive, and complete for $\mathrm{P}^{\mathrm{NEXP}}$ when KB is a stratified or general dl-program.*

8. Implementation

As stressed in the introduction, dl-programs treat DL knowledge bases and logic programs as separated modules. As a beneficial side effect of this approach, only interfacing details between the two worlds have to be known as far as an efficient implementation is concerned. This allows us to design a reasoning framework on top of existing reasoners for ASP resp. DLs. Our idea behind the implementation principle was thus to design a reasoning framework on top of existing reasoners for answer set programs resp. description logics instead of creating everything from scratch. Existing engines for both worlds have been professionally developed and are supposedly highly efficient: this, of course, does not imply that a naive coupling of state-of-the-art reasoners would perform better than a system especially tailored at evaluating dl-programs. Indeed, specific, non-naive, optimization techniques are needed for implementing this kind of coupling. This approach is also preferable, if one considers the significant manpower needed for building and releasing a system of such a complexity from scratch.

In this section, we present the architecture of our system prototype and some basic results and techniques that we used for performance improvement. A more detailed description including the algorithms can be found in [95].

8.1. System architecture

The architecture of our system prototype NLP-DL, which has been described in [28,29], is depicted in Fig. 2. The system comprises different modules, each of which is coded in the PHP scripting language; the overhead compared to a language

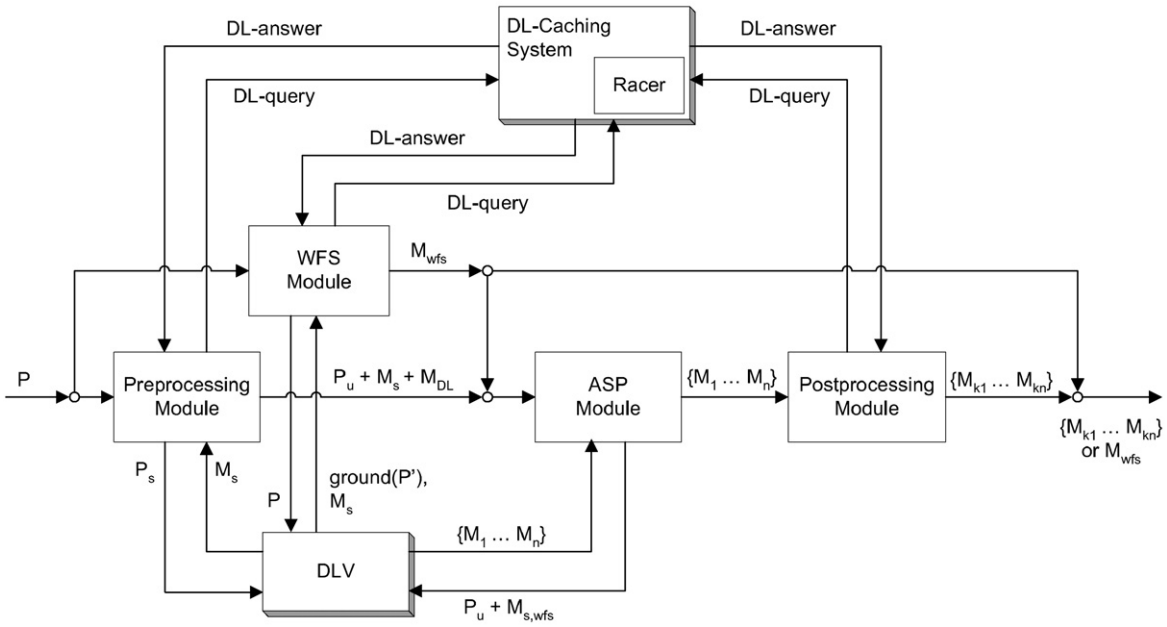

Fig. 2. System architecture of the dl-program evaluation prototype.

like C++ is insignificant, given that most of the computing power is devoted to the execution of the two external reasoners. Moreover, the choice of this language enabled us to make the prototype easily accessible by a Web-interface, thus serving its main purposes as a testing and demonstration tool. The Web-interface[11] allows the user to enter a dl-program *KB* in form of an OWL-ontology L and a program P. It can then be used either to compute model(s) or perform reasoning, both according to the selected semantics, which can be chosen between the strong answer set semantics and the well-founded semantics. The query operation mode requires the specification of one or more query atoms as input from the user; here, another choice between brave and cautious reasoning is available. Furthermore, the result can be filtered by specific predicate names.

The shadowed boxes represent the external reasoning engines: DLV [65] was used as answer set solver and RACER [45] as DL reasoner, which is embedded in a caching module.

Our prototypical implementation is capable of evaluating a dl-program in three different modes: (1) under the answer set semantics, (2) under the well-founded semantics (WFS) [33], and (3) under the answer set semantics with preliminary computation of the WFS.

The preprocessing module evaluates all dl-atoms with no input (producing the set M_{DL}), applies the splitting method (Section 8.2) (separating P into an unstratified subprogram P_u, and a stratified program P_s). The single answer set M_s of P_s is computed and returned together with P_u and M_{DL}. The ASP module implements the evaluation of P_u (which can have multiple answers $\{M_1 \dots M_n\}$), using DLV. This result is streamed to a post-processing module, which carries out the verification of each incoming answer set according to the strong answer set semantics, returning the final result $\{M_{k1} \dots M_{kn}\}$.

The WFS module is used for computing the *well-founded model* M_{wfs} of the dl-program.

The well-founded semantics is an alternative nonmonotonic semantics for logic programs [100]. In particular, it retains uniqueness of the answer to a program (called the *well-founded* model), which might be a desirable property. Also, the well-founded model approximates the intersection of all the answer sets. In [33], we extended the canonical well-founded semantics of logic programs to dl-programs, retaining many of its attractive properties. In particular, atoms which are true (resp., false) in the well-founded model, are true (resp., false) in the intersection of all the strong answer sets, provided the dl-program at hand is consistent. Thus, preliminary computation of the well-founded semantics can be exploited to reduce the size of the dl-program on which the answer set semantics has to be computed [29].

Several optimization techniques of interest were adopted, which are described in detail next.

8.2. Splitting the input program

In Section 4, we presented a method for evaluating a general dl-program. It is evident that in practice the guessing part of this algorithm generates many answer set candidates. However, when looking at the corresponding dependency graph,

[11] The prototype is accessible at http://www.kr.tuwien.ac.at/research/nlpdl.

programs are often structured in two hierarchic layers. A first, stratified layer at the bottom performs some preprocessing on the input data; under strong answer semantics there will be a single answer set. A second, unstratified layer usually is aimed at encoding some nondeterministic choice and verification. As for computing strong answer sets, it is thus desirable to constrain the usage of the general guess-and-check method to the upper layer of the program and evaluate the lower layer using a more efficient fixpoint computation. This requires that we can split the program and evaluate each part separately; this is in fact feasible, relying on the notion of a *splitting set* for programs under the answer set semantics [69]. For simplicity, our approach is to split the program only in two parts, having a fast routine for finding the strong answer set of the lower layer, while the remaining subprogram will be solved by the guess-and-check method.

Lifschitz and Turner [69] have shown that the computation of the answer sets of a logic program can be simplified by dividing the program P into two parts. Informally, first identify the unstratified subprograms of P, i.e., rules of P that contain negated cycles. Then, remove these rules from P as well as all rules that depend on P, leaving a stratified subprogram on the "bottom" of the dependency graph of P. The model of this part can now be solved by a fixpoint iteration, i.e., resulting in a unique least model. Subsequently, this model is added as extensional knowledge to the remaining, unstratified part of P, which is eventually solved by means of a guess-and-check procedure.

Formally, a *splitting set* was defined in [69] as a set U of literals such that, for every rule $r \in P$, if $H(r) \cap U \neq \emptyset$ then $lit(r) \subseteq U$, where $lit(r)$ denotes $H(r) \cup B^+(r) \cup B^-(r)$. Since in dl-programs, not only the dependency between rule body and rule head, but also between dl-atoms and their input predicates needs to be taken into account, we need to modify the definition of a splitting set. To this end, we first formalize the notion of *dependency*, which takes the occurrence of dl-atoms into account.

Definition 8.1. Let $KB = (L, P)$ be a dl-program and a and b literals occurring in some rule of P. Then, *a depends positively on b*, denoted $a \rightarrow_p b$, if one of the following conditions holds:

(P_1) There is some rule $r \in P$ such that $a \in H(r)$ and $b \in B^+(r)$.
(P_2) There are some rules $r_1, r_2 \in P$ such that $a \in B(r_1)$ and $b \in H(r_2)$ and a and b can be unified.
(P_3) There are some rules $r_1, r_2 \in P$ such that $a \in B(r_1)$ is a dl-atom, $b \in H(r_2)$ is a positive literal, and the predicate symbol of b occurs in the input list of a.

We say that *a depends negatively on b*, denoted $a \rightarrow_n b$, if one of the following conditions holds:

(N_1) There is some rule $r \in P$ such that $a \in H(r)$ and $b \in B^-(r)$.
(N_2) There is some rule $r \in P$ such that $a \in H(r)$, $b \in B(r)$ and b is a (possibly) nonmonotonic dl-atom.

The relation $\rightarrow$ is the union of $\rightarrow_p$ and $\rightarrow_n$, and $\rightarrow^+$ its transitive closure.

Example 8.2. Consider the following group of rules:

$$r_1 : p(X) \leftarrow q(X), r(X);$$

$$r_2 : q(Y) \leftarrow s(Y);$$

$$r_3 : p(X) \leftarrow DL[Student \uplus s; Person](X);$$

$$r_4 : s(X) \leftarrow enrolled(X);$$

$$r_5 : flies(X) \leftarrow bird(X), not\ penguin(X);$$

$$r_6 : part(X) \leftarrow DL[P \cap known; P](X).$$

For r_1, we have the dependencies $p(X) \rightarrow_p q(X)$ and $p(X) \rightarrow_p r(X)$, according to (P_1). Furthermore, if one considers r_1 and r_2, condition (P_2) yields $q(X) \rightarrow_p q(Y)$. For r_3 and r_4, in view of (P_3), $DL[Student \uplus s; Person](X) \rightarrow_p s(X)$ holds. As for negative dependencies, rule r_5 entails $flies(X) \rightarrow_n penguin(X)$ in view of (N_1). Finally, for rule r_6, we get $part(X) \rightarrow_n DL[P \cap known; P](X)$ according to condition (N_2).

We now define splitting sets as follows.

Definition 8.3. A *splitting set for a dl-program $KB = (L, P)$* is any set U of literals such that, for any $a \in U$, if $a \rightarrow b$, then $b \in U$. The set of rules $r \in P$ such that $H(r) \in U$ is called the *bottom of P relative to the splitting set U* and is denoted by $b_U(P)$.

To describe a method how to use this splitting for the computation of answer sets, we first need to define the notion of a *solution to KB with respect to U*, which corresponds directly to the respective notion of Lifschitz and Turner [69]. We consider a set U of literals, a set X of ground literals, and a dl-program $KB = (L, P)$. Let $ground(U)$ denote the set of all grounded literals in U. For each rule $r \in ground(P)$ such that $B^+(r) \cap ground(U) \subseteq X$ and $B^-(r) \cap ground(U)$ is disjoint from

X, create a new rule r', with $H(r') = H(r)$, $B^+(r') = B^+(r) \setminus ground(U)$ and $B^-(r') = B^-(r) \setminus ground(U)$. The ground program consisting of all such rules r' is denoted by $e_U(P, X)$.

Definition 8.4. Let U be a splitting set for a program $KB = (L, P)$. We call a pair $\langle X, Y \rangle$ of sets of ground literals a *solution* to KB with respect to U, if

- X is a strong answer set for $b_U(P)$,
- Y is a strong answer set for $e_U(P \setminus b_U(P), X \cup \{a \in ground(U) \mid a \text{ is a dl-atom and } X \models_L a\})$,
- and $X \cup Y$ is consistent.

Theorem 8.5. *Let U be a splitting set for a dl-program $KB = (L, P)$. Then, A is a strong answer set of KB iff $A = X \cup Y$ for some solution $\langle X, Y \rangle$ to KB with respect to U.*

Our aim is to find the largest subprogram of P which does not involve cycles through default negation or nonmonotonic dl-atoms.

Theorem 8.6. *Given $KB = (L, P)$, let V be the least set of literals such that (i) $a, b \in V$ whenever $a \rightarrow_n b$ and $b \rightarrow^+ a$ holds in P, and (ii) if $a \rightarrow b$ and $b \in V$, then $a \in V$. Then, the set $S = lit(P) \setminus V$ is a splitting set for P, where $lit(P) = \bigcup_{r \in P} H(r) \cup B(r)$. Furthermore, $b_S(P)$ has a single strong answer set (if it is consistent).*

In fact, if $b_S(P)$ contains only monotonic dl-atoms, then $b_S(P)$ is stratified. Moreover, any nonmonotonic dl-atom $DL[\lambda; Q](\mathbf{t})$ in $b_S(P)$ can be replaced with a monotonic dl-atom $DL[\lambda'; Q](\mathbf{t})$ where each $S_i \cap p_i$ in λ is replaced with $S_i \uplus \overline{p}_i$, where $\overline{p}_i$ is a fresh predicate, and the rule $\overline{p}_i(\vec{X}) \leftarrow not\ p_i(\vec{X})$ is added. The resulting program is stratified and has the same strong answer sets as $b_S(P)$ with respect to the original set of predicates.

Thus, in essence $b_S(P)$ is stratified. We therefore call a splitting set S as in Theorem 8.6 a *stratification splitting set* for a dl-program KB. The following property is an immediate consequence of Theorem 8.6.

Corollary 8.7. *Each dl-program has exactly one stratification splitting set.*

Example 8.8. Consider the reviewer selection program from Example 4.1. The stratification splitting set of this program comprises all literals except those with the predicates *assign*, *a*, and *error*. Thus, it has the following stratified subprogram:

$$paper(p_1);$$

$$kw(p_1, Semantic\_Web);$$

$$paper(p_2);$$

$$kw(p_2, Bioinformatics);$$

$$kw(p_2, Answer\_Set\_Programming);$$

$$kw(P, K_2) \leftarrow kw(P, K_1), DL[contains](S, K_1), DL[contains](S, K_2);$$

$$paperArea(P, A) \leftarrow DL[keywords \uplus kw; inArea](P, A);$$

$$cand(X, P) \leftarrow paperArea(P, A), DL[Referee](X), DL[expert](X, A).$$

This program is positive, and thus can only have a single strong answer set. The unstratified part of the program are the remaining rules:

$$assign(X, P) \leftarrow cand(X, P), not\ \neg assign(X, P);$$

$$\neg assign(Y, P) \leftarrow cand(Y, P), assign(X, P), X \neq Y;$$

$$a(P) \leftarrow assign(X, P);$$

$$error(P) \leftarrow paper(P), not\ a(P).$$

It follows directly from Theorem 8.5 that the strong answer sets of a dl-program $KB = (L, P)$ can be obtained by computing the unique strong answer set M of $b_U(P)$ (where U is the stratification splitting set) and then computing the strong answer sets of $e_U(P \setminus b_U(P), M')$, where M' is M augmented with the dl-atoms from $ground(U)$ which are true with respect to M. To this end, our implementation uses a fixpoint algorithm (which takes as input a stratified dl-program KB) based on results given by Theorem 5.6 and [29], in order to compute M. Then, $e_U(P \setminus b_U(P), M')$ is evaluated taking advantage of an algorithm guess (which takes as input a generic knowledge base KB), which is based on Theorem 5.8.

It is reasonable to expect that this method of splitting the dl-program is more efficient than the pure guess-and-check approach, since the "preliminary" computation of any stratified subprogram will in general narrow the search space of the guessing. A subsequent, more fine grained splitting into strongly and weakly connected components of the program will further optimize the computation. Efforts towards such a more sophisticated processing of the program's dependency information were eventually put into dlvhex, the reasoner for HEX-programs [32].

We finally remark that an analog splitting result as in Theorem 8.5 also holds for weak answer sets (the proof is similar), and similarly Theorem 8.6 except that $b_S(P)$ may have multiple weak answer sets; to ensure a single weak answer set, a finer grained stratification splitting set would be needed which addresses positive recursion through dl-atoms.

8.3. Efficient dl-atom evaluation and caching

Since the calls to the DL-reasoner are a bottleneck in the coupling of an ASP solver with a DL-engine, special methods need to be devised in order to save on the number of calls to the DL-engine. To this end, we use several complementary techniques.

8.3.1. DL-function calls

One of the features of DL-reasoners, which may be fruitfully exploited for speed up, are *non-ground* queries. In principle, when several different ground instances $a(\mathbf{c}_1), a(\mathbf{c}_2), \ldots, a(\mathbf{c}_k)$ of the dl-atom $a(\mathbf{t})$ must be evaluated, one should issue a number of separate *Boolean* calls to the DL-reasoner, in order to check whether $a(\mathbf{c}_i)$ holds, for each i. Access to the DL-reasoner has usually a non-negligible cost. Nonetheless, RACER, as well as most solvers, provides the possibility to retrieve in a function call all instances of a concept C (resp., of a role R) that are provable in the DL knowledge base . We call this reasoning task a *non-Boolean* (or *non-ground*) DL-call, meaning the possibility to ask a reasoning service at once for all the ground arguments $\mathbf{c}$ such that some dl-atom $a(\mathbf{c})$ holds.

There are cases when the set of values for $a(\mathbf{c})$ is presumably large, but we are interested in a small subset of possible values.[12] We can constrain the DL-engine to these values as follows. Assume the concept to be queried is C. We add to L axioms to the effect that $C'' \equiv C \sqcap C'$, where C' and C'' are fresh concept names, and axioms $C'(\mathbf{c}_1), \ldots, C'(\mathbf{c}_k)$; then we ask for all instances of C''. For roles, a similar but more involved approximation method is introduced, given that $\mathcal{SHIF}(\mathbf{D})$and $\mathcal{SHOIN}(\mathbf{D})$do not offer role intersection.

With the above techniques, the number of calls to the DL-reasoner can be greatly reduced. Another very useful technique to achieve this goal is caching, described next.

8.3.2. DL-caching

Whatever semantics is considered, a number of calls will be made to the DL-engine. Therefore, it is very important to avoid an unnecessary flow of data between the two engines, and to save time when a redundant DL-query has to be made. In order to achieve these objectives, it is important to introduce special caching data structures tailored for fast access to previous query calls. Such a caching system needs to deal with the case of Boolean as well as non-Boolean DL-calls.

For any dl-atom $DL[\lambda; Q](\mathbf{t})$, where $\lambda = S_1 op_1 p_1, \ldots, S_n op_n p_n$, and interpretation I, let us denote by I^λ the projection of I on $p_1, \ldots, p_n$.

Boolean DL-calls. In this case, an external call must be issued in order to verify whether a given ground dl-atom b fulfills $I \models_L b$, where I is the current interpretation and L is the DL knowledge base hosted by the DL-engine. In this setting, the caching system exploits properties of monotonic dl-atoms $a = DL[\lambda; Q](\mathbf{c})$.

Given two interpretations I_1 and I_2 such that $I_1 \subseteq I_2$, monotonicity of a implies that (i) if $I_1 \models_L a$ then $I_2 \models_L a$, and (ii) if $I_2 \not\models_L a$ then $I_1 \not\models_L a$. This property allows to set up a caching machinery where only the outcome for ground dl-atoms with minimal/maximal input is stored.

Roughly speaking, for each monotonic ground dl-atom a, we store a set $cache(a)$ of pairs $\langle I^\lambda, v\rangle$, where $v \in \{true, undefined\}$. If $\langle I^\lambda, true\rangle \in cache(a)$, then we can conclude that $J \models_L a$ for each J such that $I^\lambda \subseteq J^\lambda$. Dually, if $\langle I^\lambda, undefined\rangle \in cache(a)$, we can conclude that $J \not\models_L a$ for each J such that $I^\lambda \supseteq J^\lambda$.

We sketch the maintenance strategy for $cache(a)$ in the following. The rationale is to cache minimal (resp., maximal) input sets I^λ for which a is evaluated to *true* (resp., *undefined*) in past external calls.

Suppose a ground dl-atom $a = DL[\lambda; Q](\mathbf{c})$, an interpretation I, and a cache set $cache(a)$ are given. With a small abuse of notation, let $I(a)$ be a function whose value is *true* iff $I \models_L a$ and *undefined* otherwise. In order to check whether $I \models_L a$, $cache(a)$ is consulted and updated as follows:

(1) Check whether $cache(a)$ contains some $\langle J, v\rangle$ such that $J \subseteq I^\lambda$ and $v = true$, or $J \supseteq I^\lambda$ and $v = undefined$. If such a J exists, conclude that $I(a) = v$.

[12] E.g., if the rule $a(X) \leftarrow c(X), DL[\lambda; C](X)$ must be evaluated and the domain of c is already known and presumably smaller than the set of X s.t. $DL[\lambda; C](X)$ holds.

Fig. 3. Integrating Ontologies and Rules by defining "safe interaction" (left) vs. "safe interfaces" (right).

(2) If no such J exists, then decide $I \models_L a$ through the external DL-engine. If $I \models_L a$, then add $\langle I^\lambda, \mathit{true}\rangle$ to $\mathit{cache}(a)$, and remove from it each pair $\langle J, \mathit{true}\rangle$ such that $I^\lambda \subset J$. Otherwise (i.e., if $I \not\models_L a$), add $\langle I^\lambda, \mathit{undefined}\rangle$ to $\mathit{cache}(a)$ and remove from it each pair $\langle J, \mathit{undefined}\rangle$ such that $I^\lambda \supset J$.

Some other implementational issues are worth mentioning. First of all, since the subsumption test between sets of atoms is a critical task, some optimization is made in order to improve cache look-up. For instance, an element count is stored for each atom set, in order to prove early that $I \not\subseteq J$ whenever $|I| > |J|$. More intelligent strategies could be envisaged in this respect. Furthermore, a standard *least recently used* (LRU) algorithm has been introduced in order to keep a fixed cache size.

Non-Boolean DL-calls. In most cases, a single non-ground query for retrieving all instances of a concept or role might be employed. Caching of such queries is also possible, but cache look-up cannot take advantage of monotonicity as in the Boolean case. For each non-ground dl-atom $a = DL[\lambda; Q](\mathbf{c})$, a set $\mathit{cache}(a)$ of pairs $\langle I^\lambda, a{\downarrow}(I^\lambda)\rangle$ is maintained, where $a{\downarrow}(I)$ is the set of all ground instances a' of a such that $I \models_L a'$. Whenever for some interpretation I, $a{\downarrow}(I)$ is needed, then $\mathit{cache}(a)$ is looked up for some pair $\langle J, a{\downarrow}(J)\rangle$ such that $I^\lambda = J$.

9. Related work

In essence, related work on combining rules and ontologies can be grouped into the following three lines of research: interaction of rules and ontologies with strict semantic separation (loose coupling); interaction of rules and ontologies with strict semantic integration (tight coupling); and reductions from DLs to ASP and/or other formalisms.

For excellent surveys that classify the numerous proposals for combining rules and ontologies, we refer the interested reader to [5,83], and for discussions of general issues that come up when combining rules and ontologies to [18,27,93].

9.1. Interaction of rules and ontologies with strict semantic separation

In this setting, a (usually nonmonotonic) language plays its role in the Rules Layer, while OWL/RDF flavors are kept separate in the Ontology Layer. The two layers only communicate via a "safe interface," but do not impose syntactic restrictions on either the rules or the ontology part (see Fig. 3).

From the Rules Layer point of view, ontologies are dealt with as an external source of information whose semantics is treated separately. Nonmonotonic reasoning and rules are allowed in a decidable setting, as well as arbitrary mixing of closed and open world reasoning. This approach typically involves special predicates in rule bodies which allow queries to a DL knowledge base , and the exchange of factual knowledge. Examples for this type of interaction are dl-programs themselves and various generalizations and extensions [30,31,70,71,73,103,106]. More concretely, HEX-programs [30,31] extend the framework of dl-programs so that multiple sources of external knowledge, with possibly different semantics, might be brought into play. Probabilistic dl-programs [70,73] extend dl-programs by probabilistic uncertainty, and similarly fuzzy dl-programs [71] by fuzzy vagueness. An extension of dl-programs to handle priorities is conceived in [106]. In [103], dl-programs are extended with a framework conceived for aligning ontologies.

Further work inspired by dl-programs is [4], which combines defeasible reasoning with description logics. Like in other work mentioned above, the considered description logic serves in [4] only as an input for the default reasoning mechanism running on top of it. Moreover, similar in spirit is also the approach of calling external description logic reasoners in the TRIPLE [96] rules engine.

9.2. Interaction of rules and ontologies with strict semantic integration

This category groups formalisms that introduce rules by adapting existing semantics for rule languages directly in the Ontology Layer. The DLP [44] fragment marks one end of this spectrum while the undecidable SWRL [56,57] approach marks the other end. Nonetheless, in between, several proposals have been put forth recently to extend expressiveness while still retaining decidability; remarkably, several of these attempts build on the stable model resp. answer set semantics. Common to these approaches are syntactic restrictions of the combined language in a way that guarantees "safe interaction" of the rules and the ontology parts of the language (see Fig. 3).

Grosof et al. [44] show how inference in a subset of the description logic $\mathcal{SHOIQ}$ can be reduced to inference in a subset of Horn programs (in which no function symbols, negations, and disjunctions are permitted), and vice versa how inference in such Horn programs can be reduced to inference in $\mathcal{SHOIQ}$. This work evolved to the Web Rule Language (WRL) proposal [3].

The works by Donini et al. [21], Levy and Rousset [66], and Rosati [90,91] are representatives of hybrid approaches in which DL knowledge bases are input sources. In detail, Donini et al. [21] introduced a combination of plain datalog (without negation and disjunction) with the description logic $\mathcal{ALC}$ called $\mathcal{AL}$-*log* . An integrated knowledge base consists of a structural component in $\mathcal{ALC}$ and a relational component in datalog, where the integration of both components lies in using concepts from the structural component as "constraints" in rule bodies of the relational component. The rules must satisfy the condition that all variables in constraints atoms in a rule must also appear in ordinary atoms in the body of the same rule; this is known as *DL-safety*. Donini et al. also present a technique for answering conjunctive queries (existentially quantified conjunctions of atoms) with such constraints, where SLD-resolution as an inference method for datalog is integrated with a method for inference in $\mathcal{ALC}$.

Levy and Rousset [66] presented a combination of Horn rules with the description logic $\mathcal{ALCNR}$, called CARIN, where in contrast to [21] also roles are allowed as constraints in rule bodies. They showed that reasoning in it is undecidable already in plain settings, and singled out two decidable syntactic fragments for the rule part: non-recursive rules and recursive but *role-safe* rules, which requires that at least one variable appearing in a role atom also appears in some atom in the body with a datalog predicate which does not occur in the consequent of rules. Motik et al. [80] adopted DL-safety like Donini et al., but permitted both concepts and roles as constraints freely in the heads and bodies of rules. They established decidability of the combination with the more expressive description logics $\mathcal{SHIQ}$(D) and $\mathcal{SHOIN}$ (cf. Section 6.3), thus getting to a decidable extension of OWL with rules.[13] Horrocks et al.'s SWRL [56,57], instead, which extends OWL by rules that violate the DL-safety restriction, is undecidable. Another approach [49] in the direction of Motik et al. shows decidability for query answering in $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ with DL-safe rules by an embedding in extended conceptual logic programming, which is a decidable extension of the answer set semantics by open domains.

Rosati's *r*-hybrid knowledge bases [90,91] combined disjunctive datalog (with classical and default negation) with $\mathcal{ALC}$ based on a generalized answer set semantics. Like Levy and Rousset [66], he allowed besides concepts also roles as constraints in rule bodies, and, similar to Donini et al. [21], DL-safety was requested. Besides satisfiability, also answering ground atomic queries was discussed, based on a combination of ordinary ASP with inference in $\mathcal{ALC}$. However, since in rule heads no ontology predicate are allowed, no direct flow of information from the rules to the ontology part was facilitated.

Rosati's recent $\mathcal{DL}+log$formalism [93,94], which builds on his previous work [90,91], is the one closest in spirit to our dl-programs. In this approach, predicates are split into *DL predicates* and into *logic program* (*datalog*) *predicates*. Rules allow arbitrary disjunction of DL and datalog atoms in the head, and conjunction in the body; furthermore, atoms with a datalog predicate can occur under negation as failure. The rules must be *datalog safe*, i.e., each variable occurring in a rule must occur in an unnegated atom in the body of that rule; this is because $\mathcal{DL}+log$uses a countably infinite set of constant symbols (which coincides with the set of individuals) and makes the standard names assumption. The interaction between DL- and datalog predicates must be *weakly safe*, i.e., each variable that occurs in a DL-atom in the head must occur in a positive datalog atom in the body of the same rule. Note that differently from usual DL-safety [80], variables may occur only in atoms with DL predicates.

Rosati introduces a new notion of model of a combined rule and ontology knowledge base. A model is defined using a two-step reduct in which, in the first step, the ontology predicates are eliminated under the open-world assumption (OWA) and, in the second step, the negated logic programming predicates are removed under the closed-world assumption (CWA). As shown by Rosati, the resulting formalism is decidable provided that containment of conjunctive queries in unions of conjunctive queries over the underlying ontology is decidable. The main differences between $\mathcal{DL}+log$ and dl-programs are the following.

- $\mathcal{DL}+log$ is a tight coupling of rules and ontologies, on the basis of single models, while dl-programs provide a loose coupling of rules and ontologies, on the basis of inference. This manifests also in different behavior for reasoning by cases. In particular, the flow of disjunctive information to and from the DL knowledge base is smoother in $\mathcal{DL}+log$. For instance, given $L = \{man \sqcup woman \equiv person,\ person(lee)\}$ and P consisting of two $\mathcal{DL}+log$ rules $p(X) \leftarrow man(X)$ and $p(X) \leftarrow woman(X)$, the atom $p(lee)$ is correctly concluded from $KB = (L, P)$. As discussed in Section 4.3.3, the same behavior can however be obtained by the dl-rule $p(X) \leftarrow DL[man \sqcup woman](X)$.
- The loose coupling, as realized in dl-programs, aims at facilitating interoperability of legacy reasoning systems and software, such as DLV and RACER, which have individual underlying assumptions about the domain, treatment of equality, possible identity of individuals etc; bridging the different worlds is up to the user. On the other hand, in $\mathcal{DL}+log$ a uniform domain, uniform treatment of equality etc. assumed such that explicit bridging between the two worlds is not necessary.
- The concept of dl-atom makes extensions of dl-programs to integrate ontologies, even in different formats, straightforward; there is no corresponding counterpart in $\mathcal{DL}+log$, instead. Indeed, the approach of dl-atoms is more flexible for mixing different reasoning modalities, such as consistency checking and logical consequence. In the realm of HEX-programs [30], almost arbitrary combinations can be conceived.

[13] It is argued in [80] that this easily extends to $\mathcal{SHOIN}(\mathbf{D})$. In fact, decidability of the DL-safe rule extension holds for any DL that allows having finitely many ground literals in knowledge bases [92].

The most recent work of Motik and Rosati [75,77] aims at combining rules and ontologies in the framework of hybrid MKNF knowledge bases, which are based on the first-order variant of Lifschitz's logic MKNF [68]. Rules are of the form

$$\mathbf{K}h_1 \vee \cdots \vee \mathbf{K}h_l \leftarrow \mathbf{K}b_1, \ldots \mathbf{K}b_m, not\ b_{m+1}, \ldots, not\ b_n,$$

where all h_i and b_j are function-free first-order atoms, and $\mathbf{K}\phi$ informally means that ϕ is known to hold under the values of the *not*-atoms. To obtain decidability, DL-safety is adopted. As discussed in [77], adding such rules to an open-world DL knowledge base is a faithful extension of both logic programming and DL (in the sense that in absence of one component, the conclusions are the original ones), and allows to put on "closed world glasses". Furthermore, [77] reports that an extension permitting both modal and non-modal atoms in rules allows to generalize both SWRL and $\mathcal{DL} + log$. However, [75] reports that our dl-programs cannot be captured using MKNF rules.

Other recent works which aim at combining rules and ontologies through uniform first-order nonmonotonic formalisms are [19,20,72]. Finally, nonmonotonic extensions of DLs (but not with rules) have been proposed in [12,22].

9.3. Reductions from description logics to ASP and/or other formalisms

Some representatives of approaches reducing description logics to logic programming are the works by Van Belleghem et al. [99], Alsaç and Baral [1,7], Swift [97], Hustadt et al. [61], and Heymans and Vermeir [50,51]. In more detail, Van Belleghem et al. [99] analyze the close relationship between description logics and open logic programs, and present a mapping of DL knowledge bases in $\mathcal{ALCN}$ to open logic programs. They also show how other description logics correspond to sublanguages of open logic programs, and they explore the computational correspondences between a typical algorithm for description logic inference and the resolution procedure for open logic programs. The works by Alsaç and Baral [1,7] and Swift [97] reduce inference in the description logic $\mathcal{ALCQI}$ to query answering from normal logic programs (with default negation, but without disjunctions and classical negations) under the answer set semantics.

The remarkable work of Hustadt et al. [61] considers $\mathcal{SHIQ}$ ontologies. They reduce consistency checking and query answering to the evaluation of a positive disjunctive datalog program. Such a program is generated after an ordinary translation of the ontology to first-order logic, followed by clever application of superposition techniques and subsequent elimination of function symbols from the resulting set of clauses. The method has been practicably adopted in the KAON2 system, whose promising experimental results are accounted in [78].

Finally, Heymans and Vermeir [50,51] present an extension of disjunctive logic programming under the answer set semantics by inverses and an infinite universe. In particular, they prove that this extension is still decidable under the assumption that the rules form a tree structure, and they show how inference in the description logic $\mathcal{SHIF}$ extended by transitive closures of roles can be simulated in it.

10. Conclusion

Towards the integration of rules and ontologies in the Semantic Web, we have presented a combination of logic programming under the answer set semantics and the description logics (DLs) $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ behind the W3C standard ontology languages OWL Lite and OWL DL, respectively. We have introduced dl-programs, which consist of a DL knowledge base L and a set of dl-rules P, which may also contain queries to L in their bodies and which permit the use of nonmonotonic negation. Such programs naturally generalize both the DL and the logic programming component. Differently from other proposals, dl-programs provide a loose integration of these components, which safely interact through well-defined interfaces. This facilitates a lean bridging of the quite diverse worlds of DLs and (nonmonotonic) logic programs, and moreover provides a clean semantical basis for a coupling of reasoning engines available from the logic programming and the DL communities.

In the spirit of logic programming, we have defined Herbrand models for dl-programs, and we have generalized many well-known concepts in logic programming to dl-programs, including least models, stratifications, and answer sets. We then have derived generalizations of major results for these concepts to dl-programs, including that satisfiable positive dl-programs have a unique least Herbrand model and that satisfiable stratified dl-programs can be associated with a unique minimal Herbrand model that is characterized through iterative least Herbrand models. As for answer sets, we have presented the notion of a strong answer set, which is based on a reduction to the least model semantics of positive dl-programs, and the notion of a weak answer set, which is based on a reduction to the least model semantics of ordinary positive logic programs.

On the computational side, we gave fixpoint characterizations for the semantics of positive and stratified dl-programs, and we have shown how to compute it by finite fixpoint iterations. We have also shown how the weak answer set semantics can be reduced to the answer set semantics of ordinary normal logic programs. Furthermore, we have briefly described a prototype implementation of dl-programs, which has been built on top of the systems DLV [65] and RACER [45], and for which a number of optimization techniques have been developed. To our knowledge, this prototype is currently the most advanced implementation of a decidable combination of nonmonotonic rules and ontologies. Furthermore, we have given a precise picture of the complexity of deciding strong and weak answer set existence for a dl-program, and of brave and cautious reasoning from a dl-program under the weak and the strong answer set semantics.

Finally, we have shown how some advanced reasoning tasks like closed-world reasoning and different forms of default reasoning on ontologies can be easily realized via dl-programs. These applications fruitfully exploit nonmonotonic negation and the inherent minimality property of answer sets. They demonstrate that dl-programs are a flexible framework for accommodating different reasoning tasks on top of existing DL knowledge bases and reasoning engines, and provide a declarative "glue" for combining different inferences. Here, in particular the possibility to talk both about provability and consistency of the (possible augmented) knowledge base is a valuable feature of dl-programs. We have also shown that DL-safe rules on ontologies [80] can be emulated on top of dl-programs. We expect that the flexibility and expressiveness of dl-programs can be beneficial for a variety of applications in the context of the Semantic Web and other fields where DLs are more and more used—the work of Wang et al. on merging ontologies [103] is one example. Also other tasks like planning, diagnosis, configuration, or information integration, where ontological knowledge should be combined with knowledge in form of rules, are possible application areas.

The concept of dl-programs which we introduced here can be extended in several directions. First of all, the coupling approach is not bound to the description logics $\mathcal{SHIF}(\mathbf{D})$ or $\mathcal{SHOIN}(\mathbf{D})$, but can in principle be deployed to any DL (under necessary constraints concerning the flow of information from the logic program to the DL knowledge base). Another extension concerns modifications of the DL knowledge base before querying. In this paper, we have considered three operators which add temporarily further axioms to the knowledge base. However, it is perfectly reasonable that the update also performs removal of axioms, and that more sophisticated update operators following methods from conditional and counterfactual reasoning are applied.

A further and no less important extension is a richer language of dl-queries to the DL knowledge base . Natural candidates for this enrichment are conjunctive queries (CQs) and unions of conjunctive queries (UCQs), which are standard in the database field. Since DLs have been proposed as an expressive data model [8,16], the interest in CQs and UCQs on DL knowledge bases is increasing [15,42,82], and (restricted forms of) such queries are supported by popular DL reasoning engines like RACER, Pellet, or KAON2. Our dl-programs can be easily extended to accommodate CQs and UCQs, as done in [26]; the nice feature is that, in our framework, such a combination remains decidable, as long as query answering to the DL knowledge base (after a virtual update of the facts part) is decidable.

Finally, another direction of extension concerns the language elements on the logic programming side. Here, an extension with disjunction in rule heads is smoothly possible [27]. This is similar for the use of default negation in rule heads, and for optimization constructs like weak constraints [65]. Other extensions concern different semantics of the rules; in [33], a well-founded semantics for dl-programs has been defined, and in [104], a semantics based on defeasible logic. Other extensions concern the consideration of probabilities [70,73], and fuzziness [71], and of rule priorities [106].

On the computational side, while the current prototype implementation incorporates several optimizations, there is a lot of room for improvements. Further optimization techniques for evaluating dl-programs need to be developed. As for deployment in a distributed environment, these algorithms have to be built on top of heterogeneous reasoners. One challenging aspect here is that such algorithms will interleave the execution of a logic programming and a DL engine. Good overall performance will very much depend on the computational characteristics of the components, which may change over time as versions improve and evolve, as well as of other factors like response and data transfer time for an underlying communication medium like the Internet. Furthermore, such algorithms should exploit structural properties of dl-programs, like splitting sets and stratifiability, to a larger extent, and aim at reducing the interfacing between the logic program and the DL engine. Here, pushing work from the logic program to the DL engine might be beneficial [26]. Besides optimization, another desirable issue would be to interface different logic programming engines and DL reasoners (currently, the DLV system and RACER are interfaced). In this way, the strengths of different reasoners may be exploited as much as possible and a powerful tool made available for developing reasoning applications in a highly declarative manner.

Acknowledgements

We are grateful to Ian Horrocks, Ulrike Sattler, and Lane A. Hemaspaandra for providing valuable information on complexity-related issues during the preparation of this paper. We also thank the reviewers of this paper and its KR 2004 preliminary version, whose constructive comments helped to improve our work, and to Minh Tran Dao for help in proof reading.

This work was partially supported by the Austrian Science Fund projects P17212-N04 and Z29-N04, by the German Research Foundation under the Heisenberg Programme, the Marie Curie Individual Fellowship HPMF-CT-2001-001286 of the EU program "Human Potential" (disclaimer: the authors are solely responsible for information communicated and the European Commission is not responsible for any views or results expressed), the EU Project REWERSE (IST-2003-506779) and the Italian National Project Interlink II04CG8AGG.

Appendix A. Proofs for Section 4

Proof of Lemma 4.2. Suppose that $I_1, I_2 \subseteq HB_P$ are models of KB, that is, $I_i \models_L r$ for every $r \in ground(P)$ and $i \in \{1,2\}$. We show that $I = I_1 \cap I_2$ is also a model of KB, that is, $I \models_L r$ for every $r \in ground(P)$. Consider any $r \in ground(P)$, and assume that $I \models_L l$ for all $l \in B^+(r) = B(r)$. That is, $I \models_L l$ for all classical literals $l \in B(r)$ and $I \models_L a$ for all dl-atoms $a \in B(r)$. Hence, $I_i \models_L l$ for all classical literals $l \in B(r)$, for every $i \in \{1,2\}$. Furthermore, since every dl-atom in $ground(P)$ is monotonic

relative to KB, it holds that $I_i \models_L a$ for all dl-atoms $a \in B(r)$, for every $i \in \{1,2\}$. Since I_1 and I_2 are models of KB, it follows that $I_i \models_L H(r)$, for every $i \in \{1,2\}$, and thus $I \models_L H(r)$. This shows that $I \models_L r$. Hence, I is a model of KB. □

Proof of Theorem 4.8. Let $\mu : HB_P \cup DL_P \to \{0,1,\ldots,k\}$ be a stratification of $KB = (L,P)$ relative to DL_P^+. Recall that M_0 is the least model (and thus, in particular, a model) of $KB_0 = (L_0, P_0)$ and, for every $i \in \{1,\ldots,k\}$, it holds that M_i is the least model (and thus, in particular, a model) of $KB_i = (L_i, P_i)$ having the property that $M_i \cap HB^\star_{P_{i-1}} = M_{i-1} \cap HB^\star_{P_{i-1}}$. It thus follows that $M_k = M_{KB}$ is a model of KB. We next show that M_k is also a minimal model of KB. Towards a contradiction, suppose that there exists a model $J \subseteq HB_P$ of KB such that $J \subset M_k$. Hence, there exists some $i \in \{0,1,\ldots,k\}$ such that $J \cap HB^\star_{P_i} \neq M_k \cap HB^\star_{P_i}$. Let j be a minimal such i. Then, J is a model of KB_j. Furthermore, if $j > 0$, then $J \cap HB^\star_{P_{j-1}} = M_k \cap HB^\star_{P_{j-1}}$. But this contradicts M_j being the least model of KB_j such that $M_j \cap HB^\star_{P_{j-1}} = M_{j-1} \cap HB^\star_{P_{j-1}}$. This shows that M_k is a minimal model of KB. □

Proof of Theorem 4.12. Let $I \subseteq HB_P$. If KB is free of dl-atoms, then $sP_L^I = P^I$. Thus, I is the least model of (L, sP_L^I) iff I is the least model of P^I. Hence, I is a strong answer set of KB iff I is an answer set of P. □

Proof of Theorem 4.13. (a) Let I be a strong answer set of KB. To show that I is also a model of KB, we have to show that $I \models_L r$ for all $r \in ground(P)$. Consider any $r \in ground(P)$. Suppose that $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$. Then, the dl-rule r' that is obtained from r by removing all the literals in $B^-(r) \cup (B^+(r) \cap DL_P^?)$ is contained in sP_L^I. Since I is the least model of (L, sP_L^I) and thus, in particular, a model of (L, sP_L^I), it follows that I is a model of r'. Since $I \models_L l$ for all $l \in B^+(r')$ and $I \not\models_L l$ for all $l \in B^-(r') = \emptyset$, it follows that $I \models_L H(r)$. This shows that $I \models_L r$. Hence, I is a model of KB.

(b) By (a), every strong answer set I of KB is a model of KB. Assume that every dl-atom in DL_P is monotonic relative to KB. We now show that then I is also a minimal model of KB. Towards a contradiction, suppose the contrary. That is, suppose that there is a model J of KB with $J \subset I$. Since J is a model of KB, it follows that J is also a model of (L, sP_L^J). Since every dl-atom in DL_P is monotonic relative to KB, it then follows that $sP_L^I \subseteq sP_L^J$. Thus, J is also a model of (L, sP_L^I). But this contradicts I being the least model of (L, sP_L^I). This shows that I is a minimal model of KB. □

Proof of Theorem 4.14. Let $KB = (L,P)$ be positive. If KB is satisfiable, then M_{KB} is defined. A strong answer set of KB is an interpretation $I \subseteq HB_P$ such that I is the least model of (L, sP_L^I). Since KB is positive, it follows that sP_L^I coincides with $ground(P)$. Hence, $I \subseteq HB_P$ is a strong answer set of KB iff $I = M_{KB}$. If KB is unsatisfiable, then KB has no model. Thus, by Theorem 4.13, KB has no strong answer set.

Now assume that KB is stratified. Let μ be a stratification of KB of length $k \geqslant 0$. Suppose that $I \subseteq HB_P$ is a strong answer set of KB. That is, I is the least model of (L, sP_L^I). Hence,

- $I \cap HB^\star_{P_0}$ is the least among all models $J \subseteq HB^\star_{P_0}$ of $(L, sP0_L^I)$, and
- if $i > 0$, then $I \cap HB^\star_{P_i}$ is the least among all models $J \subseteq HB^\star_{P_i}$ of $(L, sP_i{}_L^I)$ with $J \cap HB^\star_{P_{i-1}} = I \cap HB^\star_{P_{i-1}}$.

It thus follows that

- $I \cap HB^\star_{P_0}$ is the least among all models $J \subseteq HB^\star_{P_0}$ of KB_0, and
- if $i > 0$, then $I \cap HB^\star_{P_i}$ is the least among all models $J \subseteq HB^\star_{P_i}$ of KB_i with $J \cap HB^\star_{P_{i-1}} = I \cap HB^\star_{P_{i-1}}$.

Hence, KB is consistent, and $I = M_{KB}$. Since the above line of argumentation also holds in the converse direction, it follows that $I \subseteq HB_P$ is a strong answer set of KB iff KB is consistent and $I = M_{KB}$. □

Proof of Theorem 4.19. Let $I \subseteq HB_P$. If KB is free of dl-atoms, then $wP_L^I = P^I$. Thus, I is the least model of wP_L^I iff I is the least model of P^I. Hence, I is a weak answer set of KB iff I is an answer set of P. □

Proof of Theorem 4.20. Let $I \subseteq HB_P$ be a weak answer set of KB. To show that I is also a model of KB, we have to show that $I \models_L r$ for all $r \in ground(P)$. Consider any $r \in ground(P)$. Suppose that $I \models_L l$ for all $l \in B^+(r)$ and $I \not\models_L l$ for all $l \in B^-(r)$. Then, the dl-rule r' that is obtained from r by removing all dl-atoms in $B^+(r)$ and all literals in $B^-(r)$ is in wP_L^I. Since I is the least model of wP_L^I and thus, in particular, a model of wP_L^I, it follows that $I \models_L r'$. Since $I \models_L l$ for all $l \in B^+(r')$ and $I \not\models_L l$ for all $l \in B^-(r') = \emptyset$, it follows $I \models_L H(r')$. But $H(r') = H(r)$, so $I \models_L r$. Hence, I is a model of KB. □

Proof of Theorem 4.22. Immediate from the observation that $wP_L^I = (P_L^I)^I$. □

Proof of Theorem 4.23. Let $I \subseteq HB_P$ be a strong answer set of KB. That is, I is the least model of (L, sP_L^I). Hence, I is also a model of wP_L^I. We show that I is in fact the least model of wP_L^I. Towards a contradiction, assume the contrary. That

is, assume there exists a model $J \subset I$ of wP^I_L. Hence, J is also a model of (L, sP^I_L). But this contradicts that I is the least model of (L, sP^I_L). This shows that I is the least model of wP^I_L. That is, I is a weak answer set of KB. □

Appendix B. Proofs for Section 5

Proof of Lemma 5.1. Let $I \subseteq I' \subseteq HB_P$. Consider any $r \in ground(P)$. Then, for every classical literal $l \in B(r)$, it holds that $I \models_L l$ implies $I' \models_L l$. Furthermore, since a is monotonic relative to KB, for every dl-atom $a \in B(r)$, it holds that $I \models_L a$ implies $I' \models_L a$. This shows that $T_{KB}(I) \subseteq T_{KB}(I')$. □

Proof of Proposition 5.2. ($\Rightarrow$) Assume that $T_{KB}(I) \subseteq I \subseteq HB_P$. Suppose first that I is consistent. Then, for every $r \in ground(P)$, it holds that $I \models_L l$ for all $l \in B(r)$ implies that $I \models_L H(r)$, and thus $I \models_L r$. Hence, I is a model of KB. Suppose next that I is not consistent. Then, $T_{KB}(I) = HB_P$, and thus $I = HB_P$.

($\Leftarrow$) Suppose first that I is a model of KB. That is, $I \models_L r$ for all $r \in ground(P)$. Equivalently, $I \models_L l$ for all $l \in B(r)$ implies that $I \models_L H(r)$, for all $r \in ground(P)$. Hence, $T_{KB}(I) \subseteq I$. Suppose next that $I = HB_P$. Then, $T_{KB}(I) = HB_P = I$. □

Proof of Theorem 5.4. Since T_{KB} is monotonic and HB_P is finite, it follows that $T^i_{KB}(\emptyset)$ for $i \geqslant 0$ is an increasing sequence of sets contained in $lfp(T_{KB})$, and $T^n_{KB}(\emptyset) = T^{n+1}_{KB}(\emptyset)$ for some $n \geqslant 0$. Since $T^n_{KB}(\emptyset)$ is a fixpoint of T_{KB} that is contained in $lfp(T_{KB})$, it follows that $T^n_{KB}(\emptyset) = lfp(T_{KB})$. □

Proof of Theorem 5.6. Observe first that $M_0 = \widehat{T}^{n_0}_{KB_0}(\emptyset)$, where $n_0 \geqslant 0$ such that $\widehat{T}^{n_0}_{KB_0}(\emptyset) = \widehat{T}^{n_0+1}_{KB_0}(\emptyset)$. Since $\widehat{T}^i_{KB_0}(\emptyset) = T^j_{KB_0}(\emptyset)$ for all $j \geqslant 0$, it follows by Corollary 5.3 and Theorem 5.4 that (a) M_0 is the least model of KB_0 if KB_0 is satisfiable, and (b) $M_0 = HB_P$ if KB_0 is unsatisfiable. Observe then that for $i \geqslant 1$, it holds that $M_i = \widehat{T}^{n_i}_{KB_i}(M_{i-1})$, where $n_i \geqslant 0$ such that $\widehat{T}^{n_i}_{KB_i}(M_{i-1}) = \widehat{T}^{n_i+1}_{KB_i}(M_{i-1})$. Let $KB_i = (L_i, P_i)$, and let $KB_i{}' = (L_i, P_i')$, where P_i' is the strong dl-transform of P_i relative to L_i and M_{i-1}. Then, $\widehat{T}^j_{KB_i}(M_{i-1}) = T^j_{KB_i{}'}(\emptyset) \cup M_{i-1}$ for all $j \geqslant 0$. Hence, by Corollary 5.3 and Theorem 5.4, (a) $M_i = M_{KB_i{}'} \cup M_{i-1}$ if $KB_i{}'$ is satisfiable, and (b) $M_i = HB_P$ if $KB_i{}'$ is unsatisfiable. Equivalently, (a) M_i is the least model of KB_i with $M_i \cap HB^\star_{P_{i-1}} = M_{i-1} \cap HB^\star_{P_{i-1}}$ if such a model exists, and (b) $M_i = HB_P$ if no such model exists. In summary, $M_k \neq HB_P$ iff $M_i \neq HB_P$ for all $i \in \{0, \ldots, k\}$ iff KB is consistent. Furthermore, in this case, $M_k = M_{KB}$. □

Proof of Theorem 5.8. Let $P^\star$ be defined in the same way as P_{guess}, except that every pair of rules of the form (15) is replaced by the following two rules:

$$d_a(\mathbf{c}) \leftarrow DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m;\ Q\,](\mathbf{c});$$

$$\neg d_a(\mathbf{c}) \leftarrow not\ d_a(\mathbf{c}).$$

Then, $I \subseteq HB_P$ is a weak answer set of KB iff $I^\star$ is a weak answer set of $(L, P^\star)$, where $I^\star \subseteq HB_{P^\star} = HB_{P_{\text{guess}}}$ is obtained from I by adding (i) all $d_a(\mathbf{c})$ such that $a(\mathbf{c}) \in DL_P$ and $I \models_L a(\mathbf{c})$, and (ii) all $\neg d_a(\mathbf{c})$ such that $a(\mathbf{c}) \in DL_P$ and $I \not\models_L a(\mathbf{c})$, and conversely I is obtained from $I^\star$ by restriction to HB_P. By Theorem 4.22, the latter is equivalent to $I^\star$ being an answer set of $(L, P^{\star I^\star}_L)$, where $P^{\star I^\star}_L$ is the gl*-reduct of $P^\star$ relative to L and $I^\star$. This is in turn equivalent to $I^\star$ being an answer set of P_{guess} such that $d_a(\mathbf{c}) \in I^\star$ iff $I^\star \models_L a(\mathbf{c})$, for all $a(\mathbf{c}) \in DL_P$. In summary, $I \subseteq HB_P$ is a weak answer set of KB iff I can be completed to an answer set $I^\star \subseteq HB_{P_{\text{guess}}}$ of P_{guess} such that $d_a(\mathbf{c}) \in I^\star$ iff $I^\star \models_L a(\mathbf{c})$, for all $a(\mathbf{c}) \in DL_P$. □

Appendix C. Proofs for Section 6

Proof of Theorem 6.3. By DCA and UNA, without loss of generality, we can restrict to Herbrand interpretations of L.

(1) Let M be a strong answer set of $KB^{\text{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$. Consider the Herbrand interpretation M' of L such that for each ground atom $p(c)$, $M' \models p(c)$ iff $p^+(c) \in M$. We show that M' is a $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal model M' of L. Consider Rule (22). Since M does not satisfy its body, $L(M; \lambda')$ must be satisfiable. Let N be any Herbrand model of $L(M; \lambda')$. Since for each ground atom $p(c)$, M contains either $\overline{p}(c)$ or $p^+(c)$, by construction of $L(M; \lambda')$ it holds for each p from $\mathcal{Q} \cup \mathcal{Z}$ that $N \models \neg p(c)$ iff $\overline{p}(c) \in M$ and that $N \models p(c)$ iff $p^+(c) \in M$. Furthermore, for any $\overline{p}(c)$, $\overline{p}(c) \in M$ implies $N \models \neg p(c)$. So, N must be a model of L such that $N \leqslant_{\mathcal{P}, \mathcal{Z}} M'$. Assuming that $M' \not\leqslant_{\mathcal{P}, \mathcal{Z}} N$, we derive a contradiction. Indeed, under this assumption there exists some ground fact $p(c)$, where p is from $\mathcal{P}$, such that $N \models \neg p(c)$ but $M' \models p(c)$; equivalently, $p^+(c) \in M$. This means that Rule (19) has been applied to derive $p^+(c)$; that is, $L(M; \lambda) \models p(c)$. However, since $L(M; \lambda) \subseteq L(M; \lambda')$, N is a model of $L(M; \lambda)$. Since $N \models \neg p(c)$, it follows that $L(M; \lambda) \not\models p(c)$. This is a contradiction. Thus, $M' \leqslant_{\mathcal{P}, \mathcal{Z}} N$ holds. Since $N \leqslant_{\mathcal{P}, \mathcal{Z}} M'$ and M' and N coincide on all predicates from Z, it follows that $M' = N$. Since N was an arbitrary Herbrand model of $L(M; \lambda')$, it follows that M' is a $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal model of L.

(2) Let M' be a $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal Herbrand model of L, and define

$$M = \{ p^+(c) \mid M' \models p(c) \} \cup \{ \overline{p}(c) \mid M' \models \neg p(c) \}.$$

We show that M is a strong answer set of $KB^{\text{ECWA}}_{\langle\mathcal{P},\mathcal{Z}\rangle}$. By construction of M and the fact that M' is a model of $L(M;\lambda')$, the rule stemming from (22) is trivially satisfied by M. Next, for each ground atom $p(c)$ such that p is from $\mathcal{Q}\cup\mathcal{Z}$, by construction sP^M_L contains the fact $p^+(c)$ (emerging from Rule (21)) iff $p^+(s)\in M$. Furthermore, for each ground atom $p(c)$, sP^M_L contains the fact $\overline{p}(c)$ (emerging from Rule (18) or (20)) iff $\overline{p}(s)\in M$.

Consequently, M is the least model of sP^M_L (and thus, a strong answer set of KB) if and only if for each ground atom $p(c)$ where p is from $\mathcal{P}$, we have $p^+(c)\in M$ iff $M\models DL[\lambda;p](c)$. By definition of λ and M, M' is a model of $L(M;\lambda)$. Since M' is a $\langle\mathcal{P},\mathcal{Z}\rangle$-minimal Herbrand model of L and $L\subseteq L(M;\lambda)$, M' is also a $\langle\mathcal{P},\mathcal{Z}\rangle$-minimal Herbrand model of $L(M;\lambda)$. This means that $M'\models p(c)$ iff $L(M;\lambda)\models p(c)$. By definition of M and $M\models DL[\lambda;p](c)$, it follows that $p^+(c)\in M$ iff $M\models DL[\lambda;p](c)$. Hence, M is a strong answer set of $KB^{\text{ECWA}}_{\langle\mathcal{P},\mathcal{Z}\rangle}$. □

Proof of Theorem 6.9. 1) Let M be a strong answer set of KB^{pl}, and consider $scen(M)=L\cup\{\ell_{i,0}(\mathbf{c}_{i,0})\mid g_i(\mathbf{c}_{i,0})\in M\}$. First we show that $scen(M)$ is a scenario. By the rules of form (23), $g_i(\mathbf{c}_{i,0})$ can be in $scen(M)$ only if there is some instance $(g_i)'$ of $g_i\in H$ such that $\vec{X}_{i.0}=\mathbf{c}_{i,0}$ and $L\models pc(g_i)'$. Furthermore, $scen(M)$ is satisfiable. Indeed, if this were not the case, then the rules of form (24) would include in M all literals $\neg g_i(\mathbf{c}_{i,0})$ where $\vec{X}_{i.0}=\mathbf{c}_{i,0}$, for all $1\leqslant i\leqslant n$. Hence, no rule instance of (23) is applicable, and thus M contains no positive literals $g_i(\mathbf{c}_{i,0})$. Hence, $L(M;\lambda)=L$, which implies that L is unsatisfiable. This is a contradiction. It remains to show that $scen(M)$ is maximal.

Towards a contradiction, suppose that $S\supset scen(M)$ is a maximal scenario. Then, S contains some ground literal $\ell_{i,0}(\mathbf{c}_{i,0})\notin scen(M)$, and since S is satisfiable, we have that $scen(M)\not\models\neg\ell_{i,0}(\mathbf{c}_{i,0})$; that is, $L(M;\lambda)\not\models\neg\ell_{i,0}(\mathbf{c}_{i,0})$. Hence, $\neg g_i(\mathbf{c}_{i,0})$ is not in M, since it is not derivable by the rules of form (24). This means that the strong transform sP^M_L contains a rule $g_i(\mathbf{c}_{i,0})\leftarrow DL[\ell_{i,1}](\mathbf{c}_{i,1}),\ldots,DL[\ell_{i,k_i}](\mathbf{c}_{i,k_i})$ such that $M\models_L\ell_{i,j}(\mathbf{c}_{i,j})$, for all $1\leqslant j\leqslant k_i$. Hence, $g_i(\mathbf{c}_{i,0})\in M$, which means $\ell_{i,0}(\mathbf{c}_{i,0})\in scen(M)$. This is a contradiction, which proves maximality of $scen(M)$.

2) For the maximal scenario $L\cup S$, define

$$M=\big\{g_i(\mathbf{c}_{i,0})\mid\ell_{i,0}(\mathbf{c}_{i,0})\in S\big\}\cup\big\{\neg g_i(\mathbf{c}_{i,0})\mid L\cup S\models\neg\ell_{i,0}(\mathbf{c}_{i,0})\big\},$$

where $g_i(\mathbf{c}_{i,0})$ ranges over all instances of $g_i(\vec{X}_{i,0})$, $1\leqslant i\leqslant n$. We show that M is a strong answer set of KB^{pl}.

First, we show that M is a model of the strong transform sP^M_L. By definition of M, $L(M;\lambda)=L\cup S$ and clearly all rule instances of (24), which belong to sP^M_L, are satisfied. Furthermore, each rule in sP^M_L stemming from a rule of form (23) is satisfied: if $M\models_L DL[\ell_{i,j}](\mathbf{c}_{i,j})$, for all $1\leqslant j\leqslant k_i$, and $L\cup S\not\models\neg\ell_{i,0}(\mathbf{c}_{i,0})$, then, by maximality of $L\cup S$, $\ell_{i,0}(\mathbf{c}_{i,0})\in S$, and thus $g_i(\mathbf{c}_{i,0})\in M$ by construction. Finally, M is the least model of sP^M_L: any model $N\subseteq M$ contains $g_i(\mathbf{c}_{i,0})$ iff $g_i(\mathbf{c}_{i,0})\in M$; since thus $L(N;\lambda)=L(M;\lambda)$, N also contains $\neg g_i(\mathbf{c}_{i,0})$ iff $\neg g_i(\mathbf{c}_{i,0})\in M$. This proves that M is a strong answer set of KB^{pl}. □

Proof of Theorem 6.12. For any default $\delta=\frac{\alpha_1\wedge\cdots\wedge\alpha_k:\beta_1,\ldots,\beta_n}{\gamma}$, where all α_i are literals, let $pr(\delta)=\{\alpha_1,\ldots,\alpha_k\}$, $js^-(\delta)=\{\neg\beta_1,\ldots,\neg\beta_n\}$, and $cn(\delta)=\gamma$. For sets of formulas S and S', we denote by $conc(S,S')$ the set of conclusions $cn(\delta_i')$ of all instances δ_i' of defaults $\delta_i\in D$ such that $pr(\delta_i')\subseteq S$ and $js^-(\delta_i')\cap S'=\emptyset$. Then, we recall that by Reiter's characterization of extensions in terms of generating defaults [89], $E=Cn(L\cup conc(E,E))$ holds for each extension E of Δ.

1) Suppose $E=L(M;\lambda')$ is an extension of Δ. Define $M=\{in\_\gamma_i(\mathbf{c}_i)\mid\gamma_i(\mathbf{c}_i)\in E\}\cup\{out\_\gamma_i(\mathbf{c}_i)\mid\gamma_i(\mathbf{c}_i)\notin E\}\cup\{g_i(\mathbf{c}_i)\mid\gamma_i(\mathbf{c}_i)\in conc(E,E)\}$. We show that M is an answer set of KB^{df}.

First note that, by construction, $L(M;\lambda)\subseteq L(M;\lambda')$ and $E=Cn(L(M;\lambda'))$. Furthermore, since $L(M;\lambda)=L\cup conc(E,E)$, it follows from Reiter's lemma that $E=Cn(L(M;\lambda))$ (thus λ and λ' semantically amount to the same for M).

It is therefore easy to see that M satisfies all rules in sP^M_L. It remains to show that M is the least model of sP^M_L. Let $N\subseteq M$ be the least model of sP^M_L. Clearly, N and M coincide on all predicates in_γ_i and out_γ_i, $i=1,\ldots,n$. Let $S=Cn(L\cup\{\gamma_i(\mathbf{c})\mid g_i(\mathbf{c})\in N\})=Cn(L(N;\lambda))$ $(\subseteq E)$. Since N is a model of sP^M_L, the rules stemming from (30) imply that $\{g_i(\mathbf{c})\mid\gamma_i\in conc(S,E)\}\subseteq N$, and thus $conc(S,E)\subseteq S$; therefore, S satisfies conditions 1-3 of $\Gamma_\Delta(E)$. By minimality of $\Gamma_\Delta(E)$, it follows $S=\Gamma_\Delta(E)=E$. Since $conc(S,E)=conc(E,E)$, it follows that $M\subseteq N$. Hence, $M=N$ is the least model of sP^M_L. This proves that M is a strong answer set of KB^{df}.

2) Let M be a strong answer set of KB^{df}, and let $E=Cn(L(M;\lambda'))$. By the guessing rules (27) and (28), M contains for each ground instance $\gamma_i(\mathbf{c}_i)$ of $\gamma_i(\vec{Y}_i)$ exactly one of $in\_\gamma_i(\mathbf{c})$ and $out\_\gamma_i(\mathbf{c})$. Moreover, since $p_i\uplus in\_\gamma_i$ (respectively, $p_i\uplus in\_\gamma_i$) occurs in λ', by the rules (29) $in\_\gamma_i(\mathbf{c})$ belongs to M iff $L(M;\lambda')\models\gamma_i(\mathbf{c})$ (equivalently, $\gamma_i(\mathbf{c})\in E$). Furthermore, by the rules (31) and (32), $Cn(L(M;\lambda))=Cn(L(M;\lambda'))$ must hold.

Thus, it remains to show that $E=\Gamma_\Delta(E)$. We first show that E satisfies conditions 1–3 of $\Gamma_\Delta(E)$, which means $\Gamma_\Delta(E)\subseteq E$. Since $L(M;\lambda')$ contains L, conditions 1 and 2 are clearly satisfied. As for 3, we show that E is closed under the application of defaults, i.e., $conc(E,E)\subseteq E$. Let δ_i' be an instance of default $\delta_i\in D$ such that $pr(\delta_i')\subseteq E$ and $js^-(\delta_i')\cap E=\emptyset$. Since $L(M;\lambda)\equiv L(M;\lambda')$, we have $M\models_L DL[\lambda;\alpha_{i,j}](\mathbf{c}_{i,j})$ for each $\alpha_{i,j}(\mathbf{c}_{i,j})\in pr(\delta_i')$ and $M\not\models_L DL[\lambda';\beta_{i,j}](\mathbf{c}_{i,j})$ for each $\neg\beta_{i,j}(\mathbf{c}_{i,j})\in js^-(\delta_i')$. Hence, by the rules of form (30), M must contain $cn(\delta_i')=g_i(\mathbf{c}_i)$, which implies that $\gamma_i(\mathbf{c}_i)\in L(M;\lambda)$. Consequently, $\gamma_i(\mathbf{c}_i)\in E$. This proves $conc(E,E)\subseteq E$. Thus E, satisfies conditions 1–3 of $\Gamma_\Delta(E)$.

We finally show that $E\subseteq\Gamma_\Delta(E)$. Let N result from M by removing each atom $g_i(\mathbf{c}_i)$ such that $\gamma_i(\mathbf{c}_i)\notin conc(\Gamma_\Delta(E),E)$. Since $conc(\Gamma_\Delta(E),E)\subseteq\Gamma_\Delta(E)$, N is a model of the strong transform sP^M_L. Since M is the least model of sP^M_L, $M=N$ follows, and $\Gamma_\Delta(E)$ contains each $\gamma_i(\mathbf{c}_i)$ such that $g_i(\mathbf{c}_i)\in M$; hence, $L(M;\lambda)\subseteq\Gamma_\Delta(E)$. Since $Cn(L(M;\lambda))=Cn(L(M;\lambda'))=E$, it follows $E\subseteq\Gamma_\Delta(E)$. □

Proof of Theorem 6.19. 1) Let M be a strong answer set of KB^{dls}. Then $M \not\models_L DL[\lambda; \bot](b)$ must hold, which means that $L(M;\lambda)$ is satisfiable, i.e., has some first-order model $\mathcal{I}$. For each ground atom $p(\mathbf{c}) \in ga(P)$, M contains exactly one of $p^+(\mathbf{c})$ and $p^-(\mathbf{c})$. Therefore, we have $p(\mathbf{c}) \in L(M;\lambda)$ iff $p^+(\mathbf{c}) \in M$ and $\neg p(\mathbf{c}) \in L(M;\lambda)$ iff $p^+(\mathbf{c}) \notin M$. Since M satisfies all instances of the rules of form (37), it follows that $\mathcal{I}$ satisfies $P\downarrow$. Hence, $\mathcal{I} \models (L, P\downarrow)$. Thus by Lemma 6.18, it follows that a model $\mathcal{J}$ of (L, P) exists such that $\mathcal{I} \models p(\mathbf{c})$ iff $p^+(\mathbf{c}) \in M$, for every $p(\mathbf{c}) \in ga(P)$.

2) Let M for $\mathcal{I}$ be as described. We first show that M is a model of the strong transform $P' = sP^{dls}{}^{M}_{L}$. Clearly M satisfies each rule in P' which stems from (35) or (36). Next, since $\mathcal{I}$ satisfies each ground instance of every rule r in P. M satisfies each rule in P' which stems from (37). Finally, since L is satisfiable, from the definition of M also $L(M;\lambda)$ is satisfiable. Hence, also the rule $fail \leftarrow DL[\lambda; \bot](b)$ in P' is satisfied by M. This shows that M is a model of P'. Moreover, M is the least model of P', since $p^+(\mathbf{c})$ is in M iff $p^+(\mathbf{c}) \leftarrow$ is in P' and similarly $p^-(\mathbf{c})$ is in M iff $p^-(\mathbf{c}) \leftarrow$ is in P', and $fail \notin M$. Thus, M is a strong answer set of KB^{dls}. □

Appendix D. Encoding of conjunctions/disjunctions in default reasoning

In order to encode instantiated possible hypotheses $g' = \mu_1(\mathbf{c}_1) \wedge \cdots \wedge \mu_n(\mathbf{c}_n)$ in Poole's Default Logic, where each $\mu_i(\mathbf{c}_i)$ is a ground literal with predicates p_i, we can emulate the test whether $L \cup S \cup \{\mu_1(\mathbf{c}_1) \wedge \cdots \wedge \mu_n(\mathbf{c}_n)\}$ is satisfiable for a scenario $L \cup S$ that is represented by an update $L(M;\lambda)$, by an extended update $\lambda_{g'}$ using a dl-literal $not\ DL[\lambda_{g'}; \bot](b)$, where b is an arbitrary constant, as follows.

- We add the facts

  $$q_{\mu_1(\mathbf{c}_1)}(\mathbf{c}_1) \leftarrow, \ldots, q_{\mu_n(\mathbf{c}_n)}(\mathbf{c}_n) \leftarrow$$

  in the program, where the $q_{\mu_i(\mathbf{c}_i)}$ are new predicates;
- $\lambda_{g'}$ extends λ with $p_i op_i q_{\mu_i(\mathbf{c}_i)}$, where $op_i = \uplus$ if μ_i is positive and with $op_i = \cup$ otherwise, for all $1 \leqslant i \leqslant n$. Since the predicate $q_{\mu(\mathbf{c}_i)}$ is true in each strong answer set M for (and only for) $\mathbf{c}_i$, the update $L(M;\lambda_{g'})$ amounts to $L \cup S \cup \{\mu_1(\mathbf{c}_1), \ldots, \mu_n(\mathbf{c}_n)\}$.
- We can then reformulate the "blocking" rule (24), using a fact $g(\mathbf{c})$, where $\mathbf{c} = \mathbf{c}_1, \ldots, \mathbf{c}_n$, for each instance g' of g, to

  $$\neg g(\mathbf{c}) \leftarrow DL[\lambda_{g'}; \bot](b)$$

  and replace the 'enabling" rule (24) with n rules

  $$g_i(\mathbf{c}_i) \leftarrow DL[\ell_1](\mathbf{c}'_1), \ldots, DL[\ell_k](\mathbf{c}'_k), not\ \neg g_i(\mathbf{c}_i),$$

  for $1 \leqslant i \leqslant n$, where $pc(g)' = \ell_1(\mathbf{c}'_1) \wedge \cdots \wedge \ell_k(\mathbf{c}'_k)$; here the g_i are predicates for the literals μ_i, which are used to build the update λ as if μ_i is a single literal ℓ_0 (i.e., if r_i is μ_i's predicate, then λ contains $r_i \uplus g_i$ if μ_i is positive, otherwise $r_i \cup g_i$).

Note that for encoding all possible instances g' of g, the variables in $pc(g)$ that do not occur in g itself need not be instantiated, which reduces the number of rules.

Exploiting that, for any sentence α, $L \cup S \models \alpha$ iff $L \cup S \cup \{\neg\alpha\}$ is unsatisfiable, we can similarly encode for any instance g' the test $L \cup S \models pc(g)'$ where $pc(g)' = \alpha_1 \vee \cdots \vee \alpha_k$ is a disjunction of ground literals α_i by a dl-atom $DL[\lambda_{pc(g)'}; \bot](b)$, and use it in the "blocking rule" (24); by using multiple such atoms, we can encode any precondition $pc(g)'$ that is in conjunctive normal form.

Appendix E. Proofs for Section 7

In the proof of Theorem 7.1, we make use of the following result, which is an immediate consequence of Theorem 4.22.

Lemma E.1. *Let $KB = (L, P)$ be a stratified dl-program. Then, KB has a weak answer set M iff there exists an input I (of polynomial size) of all (polynomially many) ground dl-atoms in ground(P) such that* (i) *I complies with M, and* (ii) *M is the canonical model of the ordinary stratified program P', which is obtained from ground(P) by evaluating all dl-atoms on I and removing them.*

Proof of Theorem 7.1. We prove the upper complexity bounds for stratified and general dl-programs, and the lower bounds for positive and general dl-programs.

In the stratified case, by Theorem 4.14, KB has a strong answer set iff KB is consistent. By Theorem 5.6, the latter is equivalent to $M_k \neq HB_P$, where M_k is defined by (a sequence of) fixpoint iterations and can be computed in exponential time. Hence, deciding whether KB has a strong answer set is in EXP. As for deciding whether KB has a weak answer set, by Lemma E.1, we explore (one by one) the exponentially many possible inputs I of all dl-atoms in $ground(P)$. For each such input, computing P', that is, evaluating the dl-atoms and removing them from $ground(P)$, is feasible in exponential time. We then try to compute the canonical model M of P' by (a sequence of) fixpoint iterations, and we check compliance with

the inputs I of the dl-atoms, which can both be done in exponential time. In summary, deciding whether KB has a weak answer set is also in EXP.

In the general case, we can guess an (exponential size) interpretation $I \subseteq HB_P$ and compute the transform sP_L^I (resp., wP_L^I). Since evaluating all dl-atoms in $ground(P)$ and removing (i) all default-negated literals and dl-atoms, and (ii) all not necessarily monotonic (resp., all) other dl-atoms from $ground(P)$ is feasible in exponential time, computing the transform sP_L^I (resp., wP_L^I) is also feasible in exponential time. Since we are then left with a positive ground KB', we try to compute $M_{KB'}$ by a fixpoint iteration, and check compliance with the guessed I, which can both be done in exponential time. In summary, deciding whether KB has a strong (resp., weak) answer set can be done in nondeterministic exponential time.

Hardness for EXP of deciding answer set existence in the positive case holds by a reduction from the EXP-complete problem of deciding whether a DL knowledge base L in $\mathcal{SHIF}(\mathbf{D})$ is satisfiable, since the latter is equivalent to the positive dl-program $KB = (L, \{\neg p \leftarrow, p \leftarrow DL[; \top \sqsubseteq \bot]()\})$, where p is a fresh propositional symbol, having a strong answer set, which is by Theorems 4.14, 4.20, and 4.23 in turn equivalent to KB having a weak answer set.

Hardness for NEXP of deciding answer set existence in the general case follows immediately from Theorems 4.12 and 4.19, and the fact that deciding whether an ordinary normal program has an answer set is NEXP-complete [17]. □

For the proofs of Theorems 7.2 and 7.4, we recall the concept of a domino system, which is defined as follows. A *domino system* $\mathcal{D} = (D, H, V)$ consists of a finite nonempty set D of *tiles* and two relations $H, V \subseteq D \times D$ expressing horizontal and vertical compatibility constraints between the tiles. For positive integers s and t, and a word $w = w_0 \dots w_{n-1}$ over D of length $n \leqslant s$, we say that $\mathcal{D}$ *tiles* the torus $U(s,t) = \{0,1,\dots,s-1\} \times \{0,1,\dots,t-1\}$ with initial condition w iff there exists a mapping $\tau : U(s,t) \to D$ such that for all $(x,y) \in U(s,t)$: (i) if $\tau(x,y) = d$ and $\tau((x+1) \bmod s, y) = d'$, then $(d,d') \in H$, (ii) if $\tau(x,y) = d$ and $\tau(x,(y+1) \bmod t) = d'$, then $(d,d') \in V$, and (iii) $\tau(i,0) = w_i$ for all $i \in \{0,\dots,n\}$. Condition (i) is the *horizontal constraint*, condition (ii) is the *vertical constraint*, and condition (iii) is the *initial condition*.

In the proofs of Theorems 7.2 and 7.4, we also use the following reduction of the tilability of domino systems to the satisfiability of knowledge bases in $\mathcal{SHOIN}(\mathbf{D})$ from [98] (Lemma 5.18 and Corollary 5.22) and the subsequent characterization of strong (resp., weak) answer set existence for positive dl-programs.

Lemma E.2. *For a domino system $\mathcal{D} = (D, H, V)$ with initial conditions $w = w_0 \dots w_{n-1}$, there exist DL knowledge bases L_n, $L_{\mathcal{D}}$, L_w, concepts $C_{i,0}$, $i \in \{0,1,\dots,n-1\}$, and C_d, $d \in D$, in $\mathcal{SHOIN}(\mathbf{D})$ such that*:

- *$L_n \cup L_{\mathcal{D}} \cup L_w$ is satisfiable iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition w;*
- *L_n, $L_{\mathcal{D}}$, and L_w can be constructed in polynomial time in n from n, $\mathcal{D}$, and w, respectively, and $L_w = \{C_{i,0} \sqsubseteq C_{w_i} \mid i \in \{0,1,\dots,n-1\}\}$;*
- *in every model of $L_n \cup L_{\mathcal{D}}$, each $C_{i,0}$ contains exactly one object representing $(i,0) \in U(2^{n+1}, 2^{n+1})$, and each C_d contains all objects associated with d.*

Lemma E.3. *Let $KB = (L, P)$ be a positive dl-program. Then, KB has a strong (resp., weak) answer set iff there exists an interpretation I and a subset $S \subseteq \{a \in DL_P \mid I \not\models_L a\}$ such that the ordinary positive program $P_{I,S}$, which is obtained from $ground(P)$ by deleting each rule that contains a dl-atom $a \in S$ and all remaining dl-atoms, has a model J included in I.*

Proof. ($\Rightarrow$) Suppose that KB has a strong (resp., weak) answer set I. Then, $J = I$ is a model of $P_{I,S}$ included in I, where $S \subseteq \{a \in DL_P \mid I \not\models_L a\}$.

($\Leftarrow$) Suppose that there are I and S as described in the statement of the lemma, such that $P_{I,S}$ has a model J included in I. Clearly, J satisfies all rules in $ground(P) \setminus P_{J,S}$. Since KB is positive, all dl-atoms in KB are monotonic, and thus $P_{J,S} \subseteq P_{I,S}$. Hence, J satisfies KB, and so KB is satisfiable. By Theorem 4.14, KB has a strong answer set. By Theorem 4.23, KB has also a weak answer set. □

In the proof of Theorem 7.2, we also make use of the following result, which follows from a reduction from simple Turing machines to domino systems by Börger et al. [13] (Theorem 6.1.2), and the subsequent immediate result.

Lemma E.4. *Let M be a nondeterministic Turing machine with time- (and thus space-) bound 2^n, deciding a NEXP-complete language $\mathcal{L}(M)$ over the alphabet $\Sigma = \{0, 1, ''\ ''\}$. Then, there exists a domino system $\mathcal{D} = (D, H, V)$ and a linear-time reduction trans that takes any input $b \in \Sigma^*$ to a word $w \in D^*$ with $|b| = n = |w|$ such that M accepts b iff $\mathcal{D}$ tiles the torus $U(2^{n+1}, 2^{n+1})$ with initial condition w.*

Lemma E.5. *Given a vocabulary Φ and a dl-program $KB = (L, P)$, the number of ground dl-atoms a in $ground(P)$ is polynomial, and every such ground dl-atom a has in general exponentially many different concrete inputs I_p (that is, interpretations I_p of its input predicates $p = p_1, \dots, p_m$), but each of these concrete inputs I_p has a polynomial size. Furthermore, if KB is positive, then during the computation of the canonical model of KB by fixpoint iteration, the input of any ground dl-atom a in $ground(P)$ can increase only polynomially many times, and it thus needs to be evaluated only polynomially often.*

Proof of Theorem 7.2. We prove the upper complexity bounds for positive and general dl-programs, and the lower bounds for positive and stratified dl-programs.

The NEXP-membership in the positive case follows immediately from Lemma E.3, since a suitable I and S, as described in Lemma E.3, along with proofs $I \not\models_L a$ for all $a \in S$, can be guessed and verified in exponential time.

As for the general case, by Lemma E.5, we first guess inputs I_p for all ground dl-atoms in $ground(P)$ and evaluate them with a NEXP oracle in polynomial time. For the (monotonic) ones remaining in sP_L^I, we then also guess a chain $\emptyset = I_p^0 \subset I_p^1 \subset \cdots \subset I_p^k = I_p$ (along which their inputs will be increased in the fixpoint computation below for sP_L^I) and evaluate the dl-atoms in it in polynomial time with a NEXP oracle. We finally ask a NEXP oracle if an interpretation I exists which is the answer set of sP_L^I (resp., wP_L^I) compliant with the above inputs (and thus the valuations) of the dl-atoms and such that their inputs increase in the fixpoint computation as in the above chain. This yields the $\mathrm{NP}^{\mathrm{NEXP}} = \mathrm{P}^{\mathrm{NEXP}}$ upper bound.

Hardness for NEXP of deciding (strong or weak) answer set existence in the positive case holds by a reduction from the NEXP-complete problem of deciding whether a DL knowledge base L in $\mathcal{SHOIN}(\mathbf{D})$ is satisfiable, using the same line of argumentation as in the proof of Theorem 7.1.

Hardness for $\mathrm{P}^{\mathrm{NEXP}}$ of deciding answer set existence in the stratified case is proved by a generic reduction from Turing machines M, exploiting the NEXP-hardness proof for $\mathcal{ALCQIO}$ by Tobies [98]. Informally, the main idea behind the proof is to use a dl-atom to decide the result of the j-th oracle call made by a polynomial-time bounded M with access to a NEXP oracle, where the results of the previous oracle calls are known and input to the dl-atom. By a proper sequence of dl-atom evaluations, the result of M's computation on input v can then be obtained.

More concretely, let M be a polynomial-time bounded deterministic Turing machine with access to a NEXP oracle, and let v be an input for M. Since every oracle call can simulate M's computation on v before that call, once the results of all the previous oracle calls are known, we can assume that the input of every oracle call is given by v and the results of all the previous oracle calls. Since M's computation after all oracle calls can be simulated within an additional oracle call, we can assume that the result of the last oracle call is the result of M's computation on v. Finally, since any input to an oracle call can be enlarged by "dummy" bits, we can assume that the inputs to all oracle calls have the same length $n = 2{\cdot}(k+l)$, where k is the size of v, and $l = p(k)$ is the number of all oracle calls: We assume that the input to the $m{+}1$-th oracle call (with $m \in \{0, \ldots, l{-}1\}$) has the form

$$v_k 1 v_{k-1} 1 \ldots v_1 1 c_0 1 c_1 1 \ldots c_{m-1} 1 c_m 0 \ldots c_{l-1} 0,$$

where $v_k, v_{k-1}, \ldots, v_1$ are the symbols of v in reverse order, which are all marked as valid by a subsequent "1", $c_0, c_1, \ldots, c_{m-1}$ are the results of the previous m oracle calls, which are all marked as valid by a subsequent "1", and $c_m, \ldots, c_{l-1}$ are "dummy" bits, which are all marked as invalid by a subsequent "0".

By Lemma E.4, for the NEXP oracle M', there exists a domino system $\mathcal{D} = (D, H, V)$ and a linear-time reduction *trans* that takes any input $b \in \Sigma^*$ to a word $w = w_0 \ldots w_{n-1} \in D^*$ with $|b| = n$ such that M' accepts b iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition w. By Lemma E.2, there exist DL knowledge bases L_n, $L_{\mathcal{D}}$, and L_w, and concepts $C_{i,0}$, $i \in \{0, 1, \ldots, n-1\}$, and C_d, $d \in D$, in $\mathcal{SHOIN}(\mathbf{D})$ such that (i) $L_n \cup L_{\mathcal{D}} \cup L_w$ is satisfiable iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition w, (ii) L_n, $L_{\mathcal{D}}$, and $L_w = \{C_{i,0} \sqsubseteq C_{w_i} \mid i \in \{0, 1, \ldots, n-1\}\}$ can be constructed in polynomial time in n from n, $\mathcal{D}$, and w, respectively, and (iii) in every model of $L_n \cup L_{\mathcal{D}}$, each $C_{i,0}$ contains exactly one object representing $(i, 0) \in U(2^{n+1}, 2^{n+1})$, and each C_d contains all objects associated with d.

Let the stratified dl-program $KB = (L, P)$ now be defined as follows:

$$L = L_n \cup L_{\mathcal{D}} \cup \big\{C_{i,0} \sqcap S_{i,d} \sqsubseteq C_d \mid i \in \{0, 1, \ldots, n-1\}, d \in D\big\} \cup \big\{C_{i,0}(o_i) \mid i \in \{0, 1, \ldots, n-1\}\big\},$$

$$P = \big\{\neg b^l_{2l-2}(0) \leftarrow\big\} \cup \bigcup_{j=0}^{l} P^j,$$

where $P^j = P^j_v \cup P^j_q \cup P^j_{w \leftarrow b} \cup P^j_{s \leftarrow w}$ for every $j \in \{0, \ldots, l\}$. Informally, every set of dl-rules P^j generates the input of the $j{+}1$-th oracle call, which includes the results of the first j oracle calls. Here P^l prepares, for simplicity, the input of a "dummy" (non-happening) $l{+}1$-th oracle call which contains the result of the l-th (i.e., the last) oracle call. More concretely, the bitstring $a_{-2k} \cdots a_{2l-1}$ is the input of the $j{+}1$-th oracle call iff $b^j_{-2k}(a_{-2k}), \ldots, b^j_{2l-1}(a_{2l-1})$ are in the canonical model of KB. The components P^j_v, P^j_q, $P^j_{w \leftarrow b}$, and $P^j_{s \leftarrow w}$ of P^j, with $j \in \{0, \ldots, l\}$, are defined as follows:

(1) P^0_v writes v into the input of the first oracle call, and every P^j_v copies v into the input of the $j{+}1$-th oracle call, for $j \in \{1, \ldots, l\}$:

$$P^0_v = \big\{b^0_{-2i}(v_i) \leftarrow \mid i \in \{1, \ldots, k\}\big\} \cup \big\{b^0_{-2i+1}(1) \leftarrow \mid i \in \{1, \ldots, k\}\big\},$$

$$P^j_v = \big\{b^j_{-i}(x) \leftarrow b^{j-1}_{-i}(x) \mid i \in \{1, \ldots, 2k\}\big\}.$$

(2) P_q^0 initializes the rest of the input of the first oracle call with "dummy" bits, and every P_q^j with $j \in \{1,\ldots,l\}$ writes the result of the j-th oracle call into the input of the $j{+}1$-th oracle call and carries over all the other result and dummy bits from the input of the j-th oracle call:

$$P_q^0 = \{b_i^0(0) \leftarrow \mid i \in \{0,\ldots,2l{-}1\}\},$$
$$P_q^j = \{b_i^j(x) \leftarrow b_i^{j-1}(x) \mid i \in \{0,\ldots,2l{-}1\}, i \notin \{2j{-}2, 2j{-}1\}\} \cup$$
$$\{b_{2j-2}^j(0) \leftarrow DL[\forall i,d: S_{i,d} \uplus s_{i,d}^{j-1}; \top \sqsubseteq \bot]();\ b_{2j-2}^j(1) \leftarrow not\ b_{2j-2}^j(0);\ b_{2j-1}^j(1) \leftarrow\}.$$

(3) Every $P_{w\leftarrow b}^j$ with $j \in \{0,\ldots,l\}$ realizes the above-mentioned linear-time reduction *trans*, which transforms any input b^j of the Turing machine M into an initial condition w^j of the same length of M's domino system $\mathcal{D}$.

(4) Every $P_{s\leftarrow w}^j$ with $j \in \{0,\ldots,l\}$ transforms the initial condition w^j of $\mathcal{D}$ into an input s^j to the $j{+}1$-th dl-atom via the predicates $s_{i,d}^j$:

$$P_{s\leftarrow w}^j = \{s_{i,d}^j(o_i) \leftarrow w_i^j(d) \mid i \in \{0,1,\ldots,n-1\}, d \in D\}.$$

Observe then that M accepts v iff the last oracle call returns "yes". The latter is equivalent to $b_{2l-2}^l(1)$ being derived from KB and thus $b_{2l-2}^l(0)$ being not derived from KB, which is in turn equivalent to KB having a strong (resp., weak) answer set. In summary, M accepts v iff KB has a strong (resp., weak) answer set. □

In the proofs of Theorems 7.3 and 7.4, we use the following two immediate results, which show that cautious (resp., brave) reasoning from dl-programs under the strong or weak answer set semantics can be reduced to deciding the non-existence (resp., existence) of strong or weak answer sets, and vice versa.

Lemma E.6. *Let $KB = (L, P)$ be a dl-program, and let $l \in HB_P$. Let $KB' = (L,\ P \cup \{p \leftarrow l, \neg p \leftarrow l\})$ (resp., $KB' = (L, P \cup \{p \leftarrow not\ l, \neg p \leftarrow not\ l\})$), where p is a fresh propositional symbol. Then, l belongs to every (resp., some) strong or weak answer set of KB iff KB' has no (resp., a) strong or weak answer set.*

Lemma E.7. *Let $KB = (L, P)$ be a dl-program. Then, KB has no (resp., some) strong or weak answer set iff the classical literal p belongs to every (resp., some) strong or weak answer set of $KB' = (L, P \cup \{\neg p \leftarrow\})$ (resp., $KB' = (L, P \cup \{p \leftarrow\})$), where p is a fresh propositional symbol.*

Proof of Theorem 7.3. We prove the upper complexity bounds for stratified and general dl-programs, and the lower bounds for positive and general dl-programs.

As for the upper complexity bounds, by Lemma E.6, deciding whether l belongs to every (resp., some) strong or weak answer set of $KB = (L, P)$ can be reduced to (strong or weak) answer set non-existence (resp., existence) by adding to P two rules, which do not change KB's property of being stratified or general. By Theorem 7.1, answer set existence is in EXP in the stratified case and in NEXP in the general case. Thus, deciding whether l belongs to every (resp., some) strong or weak answer set of KB is in EXP when KB is stratified, and in co-NEXP (resp., NEXP) when KB is a general dl-program.

As for the lower complexity bounds, by Lemma E.7, answer set non-existence (resp., existence) for $KB = (L, P)$ can be reduced to cautious (resp., brave) reasoning by adding a single rule to P, which does not change KB's property of being positive or general. By Theorem 7.1, answer set existence is hard for EXP in the positive case and hard for NEXP in the general case. Thus, deciding whether l belongs to every (resp., some) strong or weak answer set of KB is hard for EXP when KB is positive and hard for co-NEXP (resp., NEXP) when KB is a general dl-program. □

Proof of Theorem 7.4. We prove the upper complexity bounds for positive and general dl-programs, and the lower bounds for positive and stratified dl-programs.

We first prove the upper complexity bounds for all above cases except for brave reasoning from positive dl-programs. By Lemma E.6, deciding whether l belongs to every (resp., some) strong or weak answer set of $KB = (L, P)$ can be reduced to the complement of answer set existence (resp., answer set existence itself) by adding to P two rules. In all cases except for brave reasoning from positive dl-programs, adding the two rules does not change KB's property of being positive or general. By Theorem 7.2, answer set existence is in NEXP in the positive case and in $\mathrm{P}^{\mathrm{NEXP}}$ in the general case. Thus, deciding whether l belongs to every strong or weak answer set of KB is in co-NEXP when KB is positive, and in $\mathrm{P}^{\mathrm{NEXP}}$ when KB is a general dl-program. Furthermore, deciding whether l belongs to some strong or weak answer set of KB is in $\mathrm{P}^{\mathrm{NEXP}}$ when KB is a general dl-program.

Membership in $\mathrm{P}^{\mathrm{NEXP}}$ of brave reasoning under the weak answer set semantics in the positive case follows, by Lemma E.6, from the membership in $\mathrm{P}^{\mathrm{NEXP}}$ of deciding weak answer set existence in the stratified case.

As for the membership in D^{exp} of brave reasoning under the strong answer set semantics in the positive case, observe first that a classical literal $l \in HB_P$ belongs to some strong answer set of the positive dl-program KB iff (i) KB has some strong answer set, and (ii) KB has no strong answer set I with $l \notin I$. By Lemma E.3, the latter is equivalent to: (i) there are

I and S as described in Lemma E.3, and (ii) there are no such I and S with $l \notin I$. As argued in the proof of Theorem 7.2, (i) and (ii) are in NEXP and co-NEXP, respectively. This shows that brave reasoning in the positive case under the strong answer set semantics is in D^{exp}.

As for the lower complexity bounds for all above cases except for brave reasoning from positive dl-programs, by Lemma E.7, answer set non-existence (resp., existence) for $KB = (L, P)$ can be reduced to cautious (resp., brave) reasoning by adding a single rule to P, which does not change KB's property of being positive or stratified. By Theorem 7.2, answer set existence is NEXP-hard in the positive case and $\mathrm{P}^{\mathrm{NEXP}}$-hard in the stratified case. Hence, deciding whether l belongs to every strong or weak answer set of KB is co-NEXP-hard when KB is positive and $\mathrm{P}^{\mathrm{NEXP}}$-hard when KB is stratified. Moreover, deciding whether l is in some strong or weak answer set of KB is $\mathrm{P}^{\mathrm{NEXP}}$-hard when KB is stratified.

Hardness for D^{exp} of brave reasoning under the strong answer set semantics in the positive case holds by a reduction from a D^{exp}-hard problem involving domino systems. More concretely, by a slight adaptation of the proof of Corollary 5.14 in [98], it can be shown that there exists a domino system $\mathcal{D} = (D, H, V)$ such that the following problem is hard for D^{exp}:

($\star$) Given two initial conditions $v = v_0 \dots v_{n-1}$ and $w = w_0 \dots w_{n-1}$ over D of length n, decide whether (1) $\mathcal{D}$ tiles the torus $U(2^{n+1}, 2^{n+1})$ with initial condition v, and (2) $\mathcal{D}$ does not tile the torus $U(2^{n+1}, 2^{n+1})$ with initial condition w.

We reduce ($\star$) to brave reasoning under the strong answer set semantics in the positive case. By Lemma E.2, for domino systems $\mathcal{D} = (D, H, V)$ and initial conditions $v = v_0 \dots v_{n-1}$ and $w = w_0 \dots w_{n-1}$, there exist DL knowledge bases L_n, $L_{\mathcal{D}}$, L_v, and L_w, and concepts $C_{i,0}$, $i \in \{0, 1, \dots, n-1\}$, and C_d, $d \in D$, in $\mathcal{SHOIN}(\mathbf{D})$ such that (i) $L_n \cup L_{\mathcal{D}} \cup L_v$ is satisfiable iff $\mathcal{D}$ tiles $U(2^{n+1}, 2^{n+1})$ with initial condition v, (ii) $L_n \cup L_{\mathcal{D}} \cup L_w$ is unsatisfiable iff $\mathcal{D}$ does not tile $U(2^{n+1}, 2^{n+1})$ with initial condition w, (iii) L_n, $L_{\mathcal{D}}$, $L_v = \{C_{i,0} \sqsubseteq C_{v_i} \mid i \in \{0, 1, \dots, n-1\}\}$, and $L_w = \{C_{i,0} \sqsubseteq C_{w_i} \mid i \in \{0, 1, \dots, n-1\}\}$ can be constructed in polynomial time in n from n, $\mathcal{D}$, v, and w, respectively, and (iv) in every model of $L_n \cup L_{\mathcal{D}}$, each $C_{i,0}$ contains exactly one object representing $(i, 0) \in U(2^{n+1}, 2^{n+1})$, and each C_d contains all objects associated with d.

Let the dl-program $KB = (L, P)$ be defined as follows:

$$L = L_n \cup L_{\mathcal{D}} \cup \big\{C_{i,0} \sqcap S_{i,d} \sqsubseteq C_d \mid i \in \{0, 1, \dots, n-1\}, d \in D\big\} \cup \big\{C_{i,0}(o_i) \mid i \in \{0, 1, \dots, n-1\}\big\},$$

$$\begin{aligned} P = {} & \big\{\neg p \leftarrow, p \leftarrow DL[\forall i, d\colon S_{i,d} \uplus s_{i,d}; \top \sqsubseteq \bot]()\big\} \cup \\ & \big\{s_{i,d}(o_i) \leftarrow \mid i \in \{0, 1, \dots, n-1\}, d \in D, v_i = d\big\} \cup \\ & \big\{q \leftarrow DL[\forall i, d\colon S_{i,d} \uplus s'_{i,d}; \top \sqsubseteq \bot]()\big\} \cup \\ & \big\{s'_{i,d}(o_i) \leftarrow \mid i \in \{0, 1, \dots, n-1\}, d \in D, w_i = d\big\}. \end{aligned}$$

Observe that the dl-program KB is positive. Furthermore, KB has a strong answer set iff (1) $L_n \cup L_{\mathcal{D}} \cup L_v$ is satisfiable, and the strong answer set of KB contains q iff (2) $L_n \cup L_{\mathcal{D}} \cup L_w$ is unsatisfiable. That is, q belongs to some strong answer set of KB iff (1) $\mathcal{D}$ tiles the torus $U(2^{n+1}, 2^{n+1})$ with initial condition v, and (2) $\mathcal{D}$ does not tile the torus $U(2^{n+1}, 2^{n+1})$ with initial condition w.

Hardness for $\mathrm{P}^{\mathrm{NEXP}}$ of brave reasoning under the weak answer set semantics in the positive case is proved by a generic reduction from Turing machines. The proof is similar to the proof of $\mathrm{P}^{\mathrm{NEXP}}$-hardness of deciding strong (resp., weak) answer set existence in the stratified case (in the proof of Theorem 7.2). The main difference that must be taken into account in the construction is that rather than deciding whether a stratified dl-program has a strong (resp., weak) answer set, we now decide whether a literal holds in some weak answer set of a positive dl-program. Intuitively, we use a set of weak answer sets for guessing the outcomes of all oracle calls, and a literal q in one of these weak answers to identify the correct guess.

More concretely, let M be a polynomial-time bounded deterministic Turing machine with access to a NEXP oracle, and let v be an input for M. Let the positive dl-program $KB = (L, P)$ be defined as the stratified dl-program $KB = (L, P)$ in the proof of Theorem 7.2, except that we now add the rule

$$q \leftarrow guess\_ok^1, \dots, guess\_ok^l, call\_ok^1, \dots, call\_ok^l, \tag{E.1}$$

and that every P_q^j, $j \in \{1, \dots, l\}$, is now defined as $P_{q,id}^j \cup P_{q,guess}^j \cup P_{q,call}^j$, where:

(1) Every $P_{q,id}^j$, $j \in \{1, \dots, l\}$, copies all the persisting results and dummy bits from the input of the j-th oracle call into the input of the $j{+}1$-th oracle call:

$$P_{q,id}^j = \big\{b_i^j(x) \leftarrow b_i^{j-1}(x) \mid i \in \{0, \dots, 2l{-}1\}, i \notin \{2j{-}2, 2j{-}1\}\big\}.$$

(2) Every $P_{q,guess}^j$, $j \in \{1, \dots, l\}$, allows for guessing the outcome of the j-th oracle call, that is, exactly one fact among $b_{2j-2}^j(0)$ and $b_{2j-2}^j(1)$. The guess is verified through the predicate $guess\_ok^j$, which should evaluate to true:

$$P^j_{q,guess} = \{b^j_{2j-2}(1) \leftarrow DL[B^j \uplus b^j_{2j-2}; B^j](1);$$
$$b^j_{2j-2}(0) \leftarrow DL[B^j \uplus b^j_{2j-2}; B^j](0);$$
$$\neg b^j_{2j-2}(1) \leftarrow b^j_{2j-2}(0); guess\_ok^j \leftarrow b^j_{2j-2}(0); guess\_ok^j \leftarrow b^j_{2j-2}(1)\}.$$

(3) Every $P^j_{q,call}$, $j \in \{1,\ldots,l\}$, allows for choosing among the two possible outcomes of the j-th oracle call exactly the one that matches the result of the actual outcome. That is, $call\_ok^j$ is true iff either (a) $b^j_{2j-2}(0)$ holds and the actual outcome is "no" or (b) $b^j_{2j-2}(1)$ holds and the actual outcome is "yes":

$$P^j_{q,call} = \{\neg b^j_{2j-2}(1) \leftarrow DL[\forall i,d: S_{i,d} \uplus s^{j-1}_{i,d}; \top \sqsubseteq \bot]();$$
$$call\_ok^j \leftarrow b^j_{2j-2}(0), DL[\forall i,d: S_{i,d} \uplus s^{j-1}_{i,d}; \top \sqsubseteq \bot]();$$
$$call\_ok^j \leftarrow b^j_{2j-2}(1); b^j_{2j-1}(1) \leftarrow\}.$$

Thus, M accepts v iff (i) the last oracle call returns "yes" and (ii) the $b^j_{2j-2}(x)$'s with $j \in \{1,\ldots,l\}$ are a correct guess that matches the actual outcomes of the oracle calls. The latter is equivalent to the existence of a weak answer set of KB that contains all $guess\_ok^j$ and $call\_ok^j$ with $j \in \{1,\ldots,l\}$, or, equivalently, that contains q. In summary, M accepts v iff q holds in some weak answer set of KB. □

Appendix F. Proofs for Section 8

Proof of Theorem 8.5. We can reformulate this theorem as follows: Let U be a splitting set for a dl-program $KB = (L, P)$. A set A of literals is a strong answer set of KB if and only if A is a strong answer set of $P \setminus b_U(P) \cup M$, where M is a strong answer set of $b_U(P)$.

($\Rightarrow$) Let A be a strong answer set for KB. Let $P' = sb_U(P)^A_L$ and let S be the least model of P'. Note that S must exist. Furthermore, since $P' \subseteq sP^A_L$, it follows that $S \subseteq A$. On the other hand, since U is a splitting set for P, we must have that $P' = sb_U(P)^S_L$. Indeed, consider the set of literals $A' = \{a \in B^-(r) \cap A$ from some ground instance r of a rule in $b_U(P)$ such that $a \in A\}$. Every literal in A' must occur in the head of a ground instance of some rule r' from P; by the definition of dependencies and of a splitting set, each such r' must belong to $b_U(P)$. Therefore, $a \in S$ must hold. Consequently, S is the least model of $sb_U(P)^S_L$, which means that S is a strong answer set of $b_U(P)$.

Furthermore, A is a strong answer set of $R = P \setminus b_U(P) \cup S$. Indeed, let M be the least model of sR^A_L. Since both M and A contain S and are models of sR^A_L, $M \subseteq A$ must hold. On the other hand, if $M \subset A$, then M would be a model of sP^A_L, since M satisfies $s(P \setminus b_U(P))^A_L$ and each rule in $sb_U(P)^A_L$. Indeed, each literal $a \in M \setminus S$ must occur in some rule head of $s(P \setminus b_U(P))^A_L$, but by dependencies and splitting cannot occur in the body of any rule in $sb_U(P)^A_L$. However, this would contradict that A is a strong answer set of KB. This shows that A is a strong answer set of $P \setminus b_U(P) \cup M$, where M is a strong answer set of $b_U(P)$.

($\Leftarrow$) Let A be a strong answer set of $R = P \setminus b_U(P) \cup M$ were M is a strong answer set for $b_U(P)$. Note that $A \supseteq M$, since all literals in M appear in R as facts. The strong transform sP^A_L is given by $sP^A_L = s(P \setminus b_U(P))^A_L \cup sb_U(P)^A_L$. Each rule in the left part of the union belongs to sR^A_L. Furthermore, the right part $sb_U(P)^A_L$ must coincide with $sb_U(P)^M_L$, since each literal in $a \in A \setminus M$ must occur in the head of a rule in sP^A_L, but by dependencies and the splitting set condition cannot occur in the body of any ground instance of any rule from $b_U(P)$. Furthermore, since $M \models_L sb_U(P)^M_L$, it follows that $A \models_L sb_U(P)^M_L$. Consequently, A is a model of sP^A_L. Moreover, A must coincide with the least model N of sP^A_L. If $N \cap M \subset M$ would hold, then $N \cap M$ would be a model of $sb_U(P)^M_L$ $(= sb_U(P)^A_L)$, which contradicts that M is a strong answer set of $b_U(P)$. On the other hand, if $N \cap M = M$ $(= A \cap M)$ but $N \subset A$, then N would be a model of sR^A_L smaller than A, which contradicts that A is a strong answer set of R. It follows $N = A$. This shows that A is a strong answer set of KB. □

Proof of Theorem 8.6. Let V and S be as described. Assume that S is not a splitting set. Then there exists some $a \in S$ and some $b \in V$ such that $a \to b$. By condition (ii), $a \in V$ holds. Since $S \cap V = \emptyset$, this is a contradiction.

To show that $b_S(P)$ has a single strong answer set (if consistent), it is not hard to see that in absence of (possibly) nonmonotonic dl-atoms, $(L, b_S(P))$ has some stratification (otherwise), literals $a, b \in S$ must exist such that $a \to_n b$ and $b \to^+ a$, which is impossible. In presence of (possibly) nonmonotonic dl-atoms, the program P can be replaced by the program P' described in the discussion after the theorem. As easily seen, P' must also be stratifiable. □

References

[1] G. Alsaç, C. Baral, Reasoning in description logics using declarative logic programming, Technical report, Department of Computer Science and Engineering, Arizona State University, 2001.

[2] A. Analyti, G. Antoniou, C.V. Damásio, G. Wagner, Stable model theory for extended RDF ontologies, in: Proceedings ISWC-2005, in: LNCS, vol. 3729, Springer, 2005, pp. 21–36.

[3] J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, R. Studer, Web Rule Language (WRL), Sept. 2005, W3C Member Submission. Available at http://www.w3.org/Submission/WRL/.
[4] G. Antoniou, Non-monotonic rule systems on top of ontology layers, in: Proceedings ISWC-2002, in: LNCS, vol. 2342, Springer, 2002, pp. 394–398.
[5] G. Antoniou, C.V. Damásio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, P.F. Patel-Schneider, Combining rules and ontologies: A survey, Technical Report IST506779/Linköping/I3-D3/D/PU/a1, Linköping University, February 2005.
[6] F. Baader, B. Hollunder, Embedding defaults into terminological knowledge representation formalisms, J. Autom. Reasoning 14 (1) (1995) 149–180.
[7] C. Baral, Knowledge Representation, Reasoning, and Declarative Problem Solving, Cambridge University Press, Cambridge, UK, 2002.
[8] D. Berardi, D. Calvanese, G. De Giacomo, Reasoning on UML class diagrams, Artificial Intelligence 168 (1–2) (2005) 70–118.
[9] T. Berners-Lee, Weaving the Web, Harper, San Francisco, CA, 1999.
[10] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American 284 (5) (2001) 34–43.
[11] H. Boley, S. Tabet, G. Wagner, Design rationale of RuleML: A markup language for semantic web rules, in: I.F. Cruz, S. Decker, J. Euzenat, D.L. McGuinness (Eds.), Proceedings First Semantic Web Working Symposium (SWWS-2001), Stanford University, 2001, pp. 381–401.
[12] P.A. Bonatti, C. Lutz, F. Wolter, Description logics with circumscription, in: Proceedings KR-2006, AAAI Press, 2006, pp. 400–410.
[13] E. Börger, E. Grädel, Y. Gurevich, The Classical Decision Problem, Springer, 2001.
[14] M. Cadoli, M. Lenzerini, The complexity of propositional closed world reasoning and circumscription, J. Comput. Syst. Sci. 48 (2) (1994) 255–310.
[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Data complexity of query answering in description logics, in: Proceedings KR-2006, AAAI Press, 2006, pp. 260–270.
[16] D. Calvanese, M. Lenzerini, D. Nardi, Description logics for conceptual data modeling, in: J. Chomicki, R. van der Meyden, G. Saake (Eds.), Logics for Emerging Applications of Databases, Springer, 2003, pp. 229–263 (Chapter 8).
[17] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[18] J. de Bruijn, T. Eiter, A. Polleres, H. Tompits, On representational issues about combinations of classical theories with nonmonotonic rules, in: Proceedings KSEM-2006, in: LNCS/LNAI, vol. 4092, Springer, 2006, pp. 1–22.
[19] J. de Bruijn, T. Eiter, A. Polleres, H. Tompits, Embedding non-ground logic programs into autoepistemic logic for knowledge base combination, in: Proceedings IJCAI-2007, AAAI Press/IJCAI, 2007, pp. 304–309.
[20] J. de Bruijn, D. Pearce, A. Polleres, A. Valverde, A logic for hybrid rules, in: Proceedings RuleML-2006, IEEE Computer Society, 2006. Available at http://2006.ruleml.org/online-proceedings/rule-integ.pdf.
[21] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, $\mathcal{AL}$-log: Integrating datalog and description logics, J. Intell. Inf. Syst. 10 (3) (1998) 227–252.
[22] F.M. Donini, D. Nardi, R. Rosati, Description logics of minimal knowledge and negation as failure, ACM Trans. Comput. Log. 3 (2) (2002) 177–225.
[23] T. Eiter, W. Faber, N. Leone, G. Pfeifer, Declarative problem-solving using the DLV system, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Kluwer Academic Publishers, 2000, pp. 79–103.
[24] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Ann. Math. Artif. Intell. 15 (3/4) (1995) 289–323.
[25] T. Eiter, G. Gottlob, H. Veith, Modular logic programming and generalized quantifiers, in: Proceedings LPNMR-1997, in: LNCS/LNAI, vol. 1265, Springer, 1997, pp. 290–309.
[26] T. Eiter, G. Ianni, T. Krennwallner, R. Schindlauer, Exploiting conjunctive queries in description logic programs, in: D. Calvanese, et al. (Eds.), Proceedings of the 20th International Workshop on Description Logics (DL-2007), CEUR Workshop Proceedings, vol. 250, CEUR-WS. org, 2007, pp. 259–266.
[27] T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, H. Tompits, Reasoning with rules and ontologies, in: P. Barahona, F. Bry, E. Franconi, N. Henze, U. Sattler (Eds.), Reasoning Web, Second International Summer School 2006, Tutorial Lectures, in: LNCS, vol. 4126, Springer, 2006, pp. 93–127.
[28] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, NLP-DL: A KR system for coupling nonmonotonic logic programs with description logics, in: 4th International Semantic Web Conference (ISWC-2005), 2005. System poster.
[29] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, Non-monotonic description logic programs: Implementation and experiments, in: Proceedings LPAR-2004, in: LNCS, vol. 3452, Springer, 2005, pp. 511–517.
[30] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, A uniform integration of higher-order reasoning and external evaluations in answer-set programming, in: Proceedings IJCAI-2005, Professional Book Center, 2005, pp. 90–96.
[31] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, Effective integration of declarative rules with external evaluations for Semantic Web reasoning, in: Proceedings ESWC-2006, in: LNCS, vol. 4011, Springer, 2006, pp. 273–287.
[32] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, Towards efficient evaluation of HEX programs, in: Proceedings NMR-2006, 2006, pp. 40–46.
[33] T. Eiter, T. Lukasiewicz, R. Schindlauer, H. Tompits, Well-founded semantics for description logic programs in the Semantic Web, in: Proceedings RuleML-2004, in: LNCS, vol. 3323, Springer, 2004, pp. 81–97.
[34] W. Faber, N. Leone, G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in: Proceedings JELIA-2004, in: LNCS/LNAI, vol. 3229, Springer, 2004, pp. 200–212.
[35] D. Fensel, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, OIL: An ontology infrastructure for the Semantic Web, IEEE Intelligent Syst. 16 (2) (2001) 38–45.
[36] D. Fensel, W. Wahlster, H. Lieberman, J. Hendler (Eds.), Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, MIT Press, 2002.
[37] M. Fitting, First-Order Logic and Automated Theorem Proving, second ed., Springer, Secaucus, NJ, USA, 1996.
[38] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proceedings ICLP/SLP-1988, MIT Press, 1988, pp. 1070–1080.
[39] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gen. Comput. 9 (1991) 365–385.
[40] M. Gelfond, H. Przymusinska, T.C. Przymusinski, The extended closed world assumption and its relationship to parallel circumscription, in: Proceedings PODS-1986, ACM Press, 1986, pp. 133–139.
[41] M. Gelfond, H. Przymusinska, T.C. Przymusinski, On the relationship between circumscription and negation as failure, Artificial Intelligence 38 (1989) 75–94.
[42] B. Glimm, I. Horrocks, C. Lutz, U. Sattler, Conjunctive query answering for the description logic $\mathcal{SHIQ}$, in: Proceedings IJCAI-2007, AAAI Press/IJCAI, 2007, pp. 399–404.
[43] G. Gottlob, Complexity results for nonmonotonic logics, J. Logic Comput. 2 (3) (1992) 397–425.
[44] B.N. Grosof, I. Horrocks, R. Volz, S. Decker, Description logic programs: Combining logic programs with description logics, in: Proceedings WWW-2003, ACM Press, 2003, pp. 48–57.
[45] V. Haarslev, R. Möller, RACER system description, in: Proceedings IJCAR-2001, in: LNCS/LNAI, vol. 2083, Springer, 2001, pp. 701–705.
[46] J. Heflin, H. Munoz-Avila, LCW-based agent planning for the Semantic Web, in: Ontologies and the Semantic Web. Papers from the 2002 AAAI Workshop WS-02-11, AAAI Press, 1998, pp. 63–70.
[47] L.A. Hemachandra, The strong exponential hierarchy collapses, J. Comput. Syst. Sci. 39 (3) (1989) 299–322.
[48] J. Hendler, D.L. McGuinness, The DARPA agent markup language, IEEE Intelligent Syst. 15 (6) (2000) 67–73.
[49] S. Heymans, D.V. Nieuwenborgh, D. Vermeir, Non-monotonic ontological and rule-based reasoning with extended conceptual logic programs, in: Proceedings ESWC-2005, in: LNCS, vol. 3532, Springer, 2005, pp. 392–407.
[50] S. Heymans, D. Vermeir, Integrating ontology languages and answer set programming, in: Proceedings DEXA Workshops 2003, IEEE Computer Society, 2003, pp. 584–588.

[51] S. Heymans, D. Vermeir, Integrating semantic web reasoning and answer set programming, in: M. de Vos, A. Provetti, (Eds.), Proceedings ASP-2003—Answer Set Programming: Advances in Theory and Implementation, CEUR Workshop Proceedings, vol. 78, CEUR-WS. org, 2003, pp. 194–208.
[52] I. Horrocks, DAML + OIL: A description logic for the Semantic Web, IEEE Bull. Technical Committee Data Engrg. 25 (1) (2002) 4–9.
[53] I. Horrocks, DAML + OIL: A reason-able Web ontology language, in: Proceedings EDBT-2002, in: LNCS, vol. 2287, Springer, 2002, pp. 2–13.
[54] I. Horrocks, P.F. Patel-Schneider, Reducing OWL entailment to description logic satisfiability, in: Proceedings ISWC-2003, in: LNCS, vol. 2870, Springer, 2003, pp. 17–29.
[55] I. Horrocks, P.F. Patel-Schneider, A proposal for an OWL rules language, in: Proceedings WWW-2004, ACM Press, 2004, pp. 723–731.
[56] I. Horrocks, P.F. Patel-Schneider, S. Bechhofer, D. Tsarkov, OWL rules: A proposal and prototype implementation, J. Web Sem. 3 (1) (2005) 23–40.
[57] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web rule language combining OWL and RuleML, May 2004, W3C Member Submission. Available at http://www.w3.org/Submission/SWRL/.
[58] I. Horrocks, P.F. Patel-Schneider, F. van Harmelen, From $\mathcal{SHIQ}$ and RDF to OWL: The making of a Web ontology language, J. Web Sem. 1 (1) (2003) 7–26.
[59] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for expressive description logics, in: Proceedings LPAR-1999, in: LNCS/LNAI, vol. 1705, Springer, 1999, pp. 161–180.
[60] J.F. Horty, Some direct theories of nonmonotonic inheritance, in: D.M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. III: Non-monotonic Reasoning and Uncertain Reasoning, Clarendon Press, Oxford, 1994, pp. 111–187.
[61] U. Hustadt, B. Motik, U. Sattler, Reducing SHIQ-description logic to disjunctive datalog programs, in: Proceedings KR-2004, AAAI Press, 2004, pp. 152–162.
[62] B. Jenner, J. Toran, Computing functions with parallel queries to NP, Theor. Comput. Sci. 141 (1995) 175–193.
[63] D.S. Johnson, A catalog of complexity classes, in: J. vanLeeuwen (Ed.), Handbook of Theoretical Computer Science, vol. A, MIT Press, Cambridge, MA, 1990, pp. 67–161 (Chapter 2).
[64] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings PODS-2002, ACM Press, 2002, pp. 233–246.
[65] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (3) (2006) 499–562.
[66] A.Y. Levy, M.-C. Rousset, Combining Horn rules and description logics in CARIN, Artificial Intelligence 104 (1–2) (1998) 165–209.
[67] V. Lifschitz, Computing circumscription, in: Proceedings IJCAI-1985, Morgan Kaufmann, 1985, pp. 121–127.
[68] V. Lifschitz, Non-monotonic databases and epistemic queries, in: Proceedings IJCAI-91, Morgan Kaufmann, 1991, pp. 381–386.
[69] V. Lifschitz, H. Turner, Splitting a logic program, in: Proceedings ICLP-1994, MIT Press, 1994, pp. 23–38.
[70] T. Lukasiewicz, Probabilistic description logic programs, in: Proceedings ECSQARU-2005, in: LNCS/LNAI, vol. 3571, Springer, 2005, pp. 737–749. Extended version in Int. J. Approx. Reasoning 45 (2) (2007) 288–307.
[71] T. Lukasiewicz, Fuzzy description logic programs under the answer set semantics for the Semantic Web, in: Proceedings RuleML-2006, IEEE Computer Society, 2006, pp. 89–96. Extended version in Fundamenta Informaticae 82 (3) (2008) 289–310.
[72] T. Lukasiewicz, A novel combination of answer set programming with description logics for the Semantic Web, in: Proceedings ESWC-2007, in: LNCS, vol. 4519, Springer, 2007, pp. 384–398.
[73] T. Lukasiewicz, Tractable probabilistic description logic programs, in: Proceedings SUM-2007, in: LNCS, vol. 4772, Springer, 2007, pp. 143–156.
[74] W. Łukaszewicz, Non-Monotonic Reasoning: Formalizations of Commonsense Reasoning, Ellis Horwood, 1990.
[75] B. Motik, I. Horrocks, R. Rosati, U. Sattler, Can OWL and logic programming live together happily ever after? in: Proceedings ISWC-2006, in: LNCS, vol. 4273, Springer, 2006, pp. 501–514.
[76] B. Motik, R. Rosati, A faithful integration of description logics with logic programming, in: Proceedings IJCAI-2007, AAAI Press/IJCAI, 2007, pp. 477–482.
[77] B. Motik, R. Rosati, A faithful integration of description logics with logic programming, in: Proceedings IJCAI-2007, pp. 477–482.
[78] B. Motik, U. Sattler, A comparison of reasoning techniques for querying large description logic ABoxes, in: Proceedings LPAR-2006, in: LNCS/LNAI, Springer, 2006.
[79] B. Motik, U. Sattler, R. Studer, Query answering for OWL-DL with rules, in: Proceedings ISWC-2004, in: LNCS, vol. 3298, Springer, 2004, pp. 549–563.
[80] B. Motik, U. Sattler, R. Studer, Query answering for OWL-DL with rules, J. Web Sem. 3 (1) (2005) 41–60.
[81] I. Niemelä, P. Simons, T. Syrjänen, Smodels: A system for answer set programming, in: Proceedings NMR-2000, 2000.
[82] M. Ortiz de la Fuente, D. Calvanese, T. Eiter, E. Franconi, Characterizing data complexity for conjunctive query answering in expressive description logics, in: Proceedings AAAI-2006, AAAI Press, 2006.
[83] J.Z. Pan, E. Franconi, S. Tessaris, G. Stamou, V. Tzouvaras, L. Serafini, I. Horrocks, B. Glimm, Specification of coordination of rule and ontology languages. Project Deliverable D2.5.1, KnowledgeWeb NoE, June 2004.
[84] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[85] A. Polleres, C. Feier, A. Harth, Rules with contextually scoped negation, in: Proceedings ESWC-2006, in: LNCS, vol. 4011, Springer, 2006, pp. 332–347.
[86] D. Poole, A logical framework for default reasoning, Artificial Intelligence 36 (1) (1988) 27–47.
[87] I. Pratt-Hartmann, Complexity of the two-variable fragment with counting quantifiers, J. Logic Language Inform. 14 (3) (2005) 369–395.
[88] R. Reiter, On closed world data bases, in: H. Gallaire, J. Minker (Eds.), Logic and Data Bases, Plenum Press, New York, 1978, pp. 55–76.
[89] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.
[90] R. Rosati, Towards expressive KR systems integrating datalog and description logics: Preliminary report, in: Proceedings DL-1999, 1999, pp. 160–164.
[91] R. Rosati, On the decidability and complexity of integrating ontologies and rules, J. Web Sem. 3 (1) (2005) 61–73.
[92] R. Rosati, Semantic and computational advantages of the safe integration of ontologies and rules, in: Proceedings PPSWR-2005, in: LNCS, vol. 3703, Springer, 2005, pp. 50–64.
[93] R. Rosati, Integrating ontologies and rules: Semantic and computational issues, in: P. Barahona, F. Bry, E. Franconi, N. Henze, U. Sattler (Eds.), Reasoning Web, in: LNCS, vol. 4126, Springer, 2006, pp. 128–151.
[94] R. Rosati, $\mathcal{DL} + log$: Tight integration of description logics and disjunctive datalog, in: Proceedings KR-2006, AAAI Press, 2006, pp. 68–78.
[95] R. Schindlauer, Answer-set programming for the Semantic Web, PhD thesis, Vienna University of Technology, Austria, 2006. Available at http://www.kr.tuwien.ac.at/staff/roman/papers/thesis.pdf.
[96] M. Sintek, S. Decker, TRIPLE—A query, inference, and transformation language for the Semantic Web, in: Proceedings ISWC-2002, in: LNCS, vol. 2342, Springer, 2002, pp. 364–378.
[97] T. Swift, Deduction in ontologies via ASP, in: Proceedings LPNMR-2004, in: LNCS/LNAI, vol. 2923, Springer, 2004, pp. 275–288.
[98] S. Tobies, Complexity results and practical algorithms for logics in knowledge representation, PhD thesis, RWTH Aachen, Germany, 2001.
[99] K. Van Belleghem, M. Denecker, D. De Schreye, A strong correspondence between description logics and open logic programming, in: Proceedings ICLP-1997, MIT Press, 1997, pp. 346–360.
[100] A. van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, Journal of the ACM 38 (3) (1991) 620–650.
[101] W3C, OWL Web ontology language overview, 2004. Available at http://www.w3.org/TR/2004/REC-owl-features-20040210/.
[102] W3C, OWL 1.1 Web ontology language overview, 2006. Available at http://www.w3.org/Submission/owl11-overview/.

[103] K. Wang, G. Antoniou, R.W. Topor, A. Sattar, Merging and aligning ontologies in dl-programs, in: Proceedings RuleML-2005, in: LNCS, vol. 3791, Springer, 2005, pp. 160–171.
[104] K. Wang, D. Billington, J. Blee, G. Antoniou, Combining description logic and defeasible logic for the Semantic Web, in: Proceedings RuleML-2004, in: LNCS, vol. 3323, Springer, 2004, pp. 170–181.
[105] S. Woltran, Answer set programming: Model applications and proofs-of-concept, Technical Report WP5, Working Group on Answer Set Programming (WASP, IST-FET-2001-37004), July 2005. Available at http://www.kr.tuwien.ac.at/projects/WASP/report.html.
[106] F. Yang, X. Chen, Z. Wang, p-dl-programs: Combining dl-programs with preference for Semantic Web. Unpublished manuscript, Aug. 2006.