I N F S Y S
R E S E A R C H
R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME
ARBEITSGRUPPE WISSENSBASIERTE SYSTEME

COMBINING ANSWER SET PROGRAMMING WITH DESCRIPTION LOGICS FOR THE SEMANTIC WEB

THOMAS EITER GIOVAMBATTISTA IANNI THOMAS LUKASIEWICZ
ROMAN SCHINDLAUER HANS TOMPITS

INFSYS RESEARCH REPORT 1843-07-04
MARCH 2007

Institut für Informationssysteme
AG Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TU WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-07-04, MARCH 2007

COMBINING ANSWER SET PROGRAMMING WITH DESCRIPTION LOGICS FOR THE SEMANTIC WEB

MARCH 14, 2007

Thomas Eiter[1] Giovambattista Ianni[2 1] Thomas Lukasiewicz[3 1]
Roman Schindlauer[1] Hans Tompits[1]

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 of *description logic rules* (or *dl-rules*) P. Such rules are similar to usual rules in nonmonotonic logic programs, but 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 reduct, 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]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria; e-mail: {eiter, ianni, lukasiewicz, roman, tompits}@kr.tuwien.ac.at.

[2]Dipartimento di Matematica, Università della Calabria, P.te P. Bucci, Cubo 30B, I-87036 Rende, Italy

[3]Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Rome, Italy; e-mail: lukasiewicz@dis.uniroma1.it.

Acknowledgements: This work was partially supported by the Austrian Science Fund projects P17212-N04 and Z29-N04, a Heisenberg Professorship of the German Research Foundation, 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), and the EU Project REWERSE (IST-2003-506779).

Contents

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 the 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 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, 37] 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, 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* [98, 59].

The language OWL provides the three increasingly expressive sublanguages, *OWL Lite*, *OWL DL*, and *OWL Full*, where OWL DL basically corresponds to the Web ontology language DAML+OIL [53, 54], which was developed by merging DAML [49] and OIL [36]. The languages OWL Lite and OWL DL are essentially very expressive Description Logics (DLs) with an RDF syntax [59]. 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 an automated reasoning support. In fact, as shown in [55], 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 [56] 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. 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* (cf. [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. [101] for a comprehensive report on 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 of *description logic rules* (or *dl-rules* for short) P. Such rules are similar to usual rules in logic programs with negation as failure, but 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. 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 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 the 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 reduct. 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 algorithms, 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 [85, 40, 41] and default reasoning [84, 86] 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 [78].

- 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 B 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 (more often called strong) negation. We first describe the syntax and then the semantics of normal logic programs.

2.1 Syntax

Roughly, a normal (logic) program is a finite set of rules, where each rule has a classical literal in its head and a conjunction of classical literals and default-negated classical literals in its body. Such a program is positive if it is free of default negations.

More concretely, let Φ be a first-order vocabulary with nonempty finite sets of constant and 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 $not\, l$. A *normal rule* (or simply *rule*) r is an expression of the form

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

where $a, b_1, \ldots, b_m$ are classical literals. The literal a is the *head* of the rule r, while 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 "$\leftarrow$". 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. We denote by $ground(P)$ the set of all 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 *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 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 formula ϕ, decide whether ϕ holds in every (resp., some) answer set of P (*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 (*answer set checking*); and (4) compute the set of all answer sets of a given program P.

3 The Description Logics $\mathcal{SHIF}(\mathrm{D})$ and $\mathcal{SHOIN}(\mathrm{D})$

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 stand behind the Web ontology languages OWL Lite and OWL DL, respectively (see [55, 59] 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 of *elementary datatypes* and a set of *data values*. A *datatype* is either an elementary datatype or a set of data values (called *datatype oneOf*). 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 $\Delta^{\mathbf{D}}$. The mapping $\cdot^{\mathbf{D}}$ is extended to all datatypes by $\{v_1, \ldots\}^{\mathbf{D}} = \{v_1^{\mathbf{D}}, \ldots\}$. Let $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ be pairwise disjoint finite nonempty 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., $a \neq b$) (called *equality* (resp., *inequality*) *axiom*), where $a, b \in \mathbf{I}$. A (*description logic*) *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, let $S \sqsubseteq^\star R$ relative to L 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 [60].

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

Example 3.1 (Scientific Publications) Suppose we want to store some knowledge about scientific publications, using a description logic knowledge base. Let *string* and $\mathbb{N}$ denote the datatypes of strings and natural numbers, respectively. Let the concepts $Publication$ and $Paper$ denote the classes of publications and (conference or journal) papers, respectively. Let the datatype roles $title$ and $year$ associate with every publication a string resp. natural number, which represent its title resp. publication year. The following axioms then encode that (1) $title$ is a binary relation between publications and strings, (2) $year$ is a binary relation between publications and natural numbers, (3) every paper is a publication, (4) pub_1 is a publication, (5) pub_1 has the title "Classical Negation in Logic Programs and Disjunctive Databases", and (6) pub_1 was published in 1991 (see Appendix A for a more detailed description logic knowledge base including scientific publications):

(1) $\geq 1\ title \sqsubseteq Publication$; $\top \sqsubseteq \forall title.string$;
(2) $\geq 1\ year \sqsubseteq Publication$; $\top \sqsubseteq \forall year.\mathbb{N}$;
(3) $Paper \sqsubseteq Publication$;
(4) $Paper(pub_1)$;
(5) $title(pub_1, \text{"}Classical\ Negation\ in\ Logic\ Programs\ and\ Disjunctive\ Databases\text{"})$;
(6) $year(pub_1, \text{"}1991\text{"})$.

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 $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
- $(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$;
- $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$;
- $(\geqslant nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x,y) \in R^{\mathcal{I}}\}) \geqslant n\}$;
- $(\leqslant nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x,y) \in R^{\mathcal{I}}\}) \leqslant n\}$;
- $(\exists U.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x,y) \in U^{\mathcal{I}} \wedge y \in D^{\mathbf{D}}\}$;
- $(\forall U.D)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x,y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathbf{D}}\}$;
- $(\geqslant nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x,y) \in U^{\mathcal{I}}\}) \geqslant n\}$;
- $(\leqslant nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \#(\{y \mid (x,y) \in U^{\mathcal{I}}\}) \leqslant n\}$.

The *satisfaction* of a description logic axiom F in the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with respect to a datatype theory $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$, 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 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 description logic knowledge bases 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 $L \models C(a)$; and (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., $L \models U(a,v)$). 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 $\mathcal{SHOIN}(\mathbf{D})$, (4) and (5) are special cases of (3).

Example 3.2 (Scientific Publications, 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 publications and the class of all papers both may have some instances. Furthermore, L logically implies all its axioms as well as the axiom $Publication(pub_1)$.

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 description logic knowledge bases under their standard first-order semantics. We first define the syntax of dl-programs and then their semantics.

4.1 Syntax

Informally, a dl-program consists of a description logic 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. Roughly, in such a query, it is asked whether a certain description logic axiom or its negation logically follows from L or not.

We first define the notions of dl-queries and dl-atoms, which are used in rule bodies to express queries to the description logic 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; or (c) of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role, and t_1, t_2 are terms. A *dl-atom* has the form

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

where each S_i is either a concept or a role, $op_i \in \{\uplus, \cup, \cap\}$, p_i is a unary resp. binary predicate symbol, 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.

We use the following example to illustrate our main ideas.

Example 4.1 (Reviewer Selection) Suppose we want to assign reviewers to papers, based on certain information about the papers and available persons, using a description logic knowledge base L_S about scientific publications (such as the one sketched in Example 3.1 and given in more detail in Appendix A).

We assume not to be aware of the entire structure and contents of L_S, but of the following aspects. The description logic knowledge base L_S classifies papers into research areas, stored in the concept *Area*, 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 (obtained, e.g., by reification of the classes). Furthermore, the abstract role *expert* relates persons to their areas of expertise, and the concept *Referee* contains all referees. Finally, the abstract role *hasMember* associates with a cluster of similar keywords all its members. Consider then the dl-program $KB_S = (L_S, P_S)$, where P_S is given by the following dl-rules:

(1) $paper(p_1)$; $kw(p_1, Semantic\_Web)$;
(2) $paper(p_2)$; $kw(p_2, Bioinformatics)$; $kw(p_2, Answer\_Set\_Programming)$;
(3) $kw(P, K_2) \leftarrow kw(P, K_1),\ DL[hasMember](S, K_1), DL[hasMember](S, K_2)$;
(4) $paperArea(P, A) \leftarrow DL[keywords \uplus kw; inArea](P, A)$;
(5) $cand(X, P) \leftarrow paperArea(P, A),\ DL[Referee](X),\ DL[expert](X, A)$;
(6) $assign(X, P) \leftarrow cand(X, P), not\ \neg assign(X, P)$;
(7) $\neg assign(Y, P) \leftarrow cand(Y, P), assign(X, P),\ X \neq Y$;
(8) $a(P) \leftarrow assign(X, P)$;
(9) $error(P) \leftarrow paper(P), not\ a(P)$.

Intuitively, lines (1) and (2) specify the keyword information of the two papers p_1 and p_2, which should be assigned to reviewers. The rule (3) augments, by choice of the designer, the keyword information with similar ones (hoping for good). The rule (4) queries the augmented L_S to retrieve the areas that each paper is classified into, and the rule (5) singles out review candidates based on this information from experts among the reviewers according to L_S. Rules (6) and (7) pick one of the candidate reviewers for a paper (multiple

reviewers can be selected similarly). Finally, (8) and (9) check if each paper is assigned; if not, an error is flagged. Note that, in view of rules (3)–(5), information flows in both directions between the description logic knowledge base L_S and the knowledge represented by the above dl-program.

To illustrate the use of the operator $\uplus$, a predicate $poss\_Referees$ may be defined in the dl-program, and "$Referee \uplus poss\_Referees$" may be added in the first dl-atom of (5), which thus constrains the set of referees.

The dl-rule below shows in particular how dl-rules can be used to encode certain qualified number restrictions, which are not available in $\mathcal{SHOIN}(\mathbf{D})$. It defines an *expert* as an author of at least three papers of the same area:

$$\begin{aligned} expert(X, A) \leftarrow\ & DL[isAuthorOf](X, P_1),\ DL[isAuthorOf](X, P_2), \\ & DL[isAuthorOf](X, P_3),\ DL[inArea](P_1, A), \\ & DL[inArea](P_2, A),\ DL[inArea](P_3, A), \\ & P_1 \neq P_2,\ P_2 \neq P_3,\ P_3 \neq P_1. \end{aligned}$$

4.2 Semantics

We now define the semantics of dl-programs. We first define Herbrand interpretations and the truth of dl-programs in Herbrand interpretations. The latter is done by defining the truth of ground dl-atoms in Herbrand interpretations. We then define a canonical least model semantics and a canonical iterative least model semantics of positive and stratified dl-programs, respectively. We finally define two alternative notions of answer sets of general dl-programs, namely *strong* and *weak answer sets*. In the sequel, let $KB = (L, P)$ be a dl-program.

The *Herbrand base* of P, denoted HB_P, is the set of all ground literals with a standard predicate symbol that occurs in P and constant symbols in Φ. An *interpretation* I relative to P is a consistent subset of HB_P. We define that I is a *model* of a ground literal or dl-atom l (or I *satisfies* l) under L, denoted $I \models_L l$, as follows:

- 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 $1 \leq i \leq m$,

$$A_i(I) = \begin{cases} \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \uplus; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \cup; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I \text{ does not hold}\}, & \text{if } op_i = \uplus. \end{cases}$$

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.

4.2.1 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 that a similar result holds for positive dl-programs KB as well.

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.*

As an immediate corollary, every satisfiable positive dl-program KB has a unique least model, denoted M_{KB}, which is contained in every model of KB.

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.*

Example 4.4 Let the dl-program KB be given by the dl-program KB_S of Example 4.1 without the rules (6)–(9). 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 contains all review candidates for the given papers p_1 and p_2.

4.2.2 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 linked via default-negation and certain dl-atoms. Like for ordinary stratified programs, a canonical minimal model 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, if we do not know a priori whether some dl-atoms are monotonic, and determining this might be costly; notice, however, that absence of $\cap$ in (2) is a simple syntactic criterion that implies monotonicity of a dl-atom (cf. also Example 4.4).

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 $a \in DL_P$ 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. A *stratification of $KB = (L, P)$ (relative to DL_P^+)* is a mapping $\lambda : HB_P \cup DL_P \to \{0, 1, \ldots, k\}$ such that

(i) for each $r \in ground(P)$, $\lambda(H(r)) \geqslant \lambda(l')$ for each $l' \in B^+(r)$, and $\lambda(H(r)) > \lambda(l')$ for each $l' \in B^-(r)$), and

(ii) $\lambda(a) \geqslant \lambda(l)$ for each input literal l of each $a \in DL_P^+$, and $\lambda(a) > \lambda(l))$ for each input literal l of each $a \in DL_P^?$.

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

A dl-program $KB = (L, P)$ is *stratified* iff it has a stratification λ of some 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 model of KB_i such that $M_i | HB^{\star}_{P_{i-1}} = M_{i-1} | HB^{\star}_{P_{i-1}}$.

We say KB is *consistent*, if every M_i with $i \in \{0, \ldots, k\}$ exists, and KB is *inconsistent* otherwise. If KB is consistent, then M_{KB} denotes M_k. Notice that M_{KB} is well-defined, since it does not depend on a particular λ (cf. Corollary 4.11). The following result shows that M_{KB} is in fact a minimal model of KB.

Theorem 4.5 *Let $KB = (L, P)$ be a stratified dl-program. Then, M_{KB} is a minimal model of KB.*

Example 4.6 Consider the dl-program $KB = (L, P)$ given by the dl-program KB_S of Example 4.1, but without the rules (6) and (7). 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.

4.2.3 Strong Answer Set Semantics of dl-Programs

We now define the *strong answer set semantics* of general dl-programs KB, 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* of KB (see Section 4.2.4).

In the sequel, 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

(i) deleting 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) deleting 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 4.7 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}).

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 4.8 *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 4.9 *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 positive resp. stratified dl-programs KB are satisfiable resp. consistent iff they have a strong answer set. In this case, they have at most one strong answer set, which coincides with their canonical minimal model M_{KB}.

Theorem 4.10 *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 λ of KB, we thus obtain that the notion of consistency of KB and the canonical minimal model M_{KB} are both independent of λ.

Corollary 4.11 *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 4.12 Consider now the full dl-program of Example 4.1. This program is not stratified, in view of the rules (6) and (7), 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.

4.2.4 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.

In the sequel, 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

(i) deleting 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) deleting 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 4.13 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.

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 4.14 *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, the weak answer sets of KB are generally not minimal models of KB, even if KB has only monotonic dl-atoms.

Theorem 4.15 *Let $KB = (L, P)$ be a dl-program. Then, every weak answer set of KB is also a model of KB.*

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

Theorem 4.16 *Let $KB = (L, P)$ be a dl-program. Let $I \subseteq HB_P$ and let P_L^I be 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 $B^+(r) \cup B^-(r)$. Then, I is a weak answer set of KB iff I is an answer set of P_L^I.*

Finally, the following 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, which have a weak answer set that is not a strong answer set.

Theorem 4.17 *Every strong answer set of a dl-program $KB = (L, P)$ is also a weak answer set of KB.*

Example 4.18 Assume it is given the short program $P = \{p(a) \leftarrow DL[c \uplus p; c](a)\}$ 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 (Theorem 4.9) the strong answer set semantics (in case all dl-atoms are known to be monotonic) does. Thus M_2, although not minimal, is a weak answer set. M_2 might be considered a counter-intuitive 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, can be found in [35].

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 4.17, 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 analog dl-programs KB. This can be exploited for a bottom-up computation of the strong answer set of KB.

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

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

The following lemma shows that for positive dl-programs 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 $\mathit{ground}(P)$ 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 $\mathit{lfp}(T_{KB})$, 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) $\mathit{lfp}(T_{KB}) = M_{KB}$, if KB is satisfiable, and (b) $\mathit{lfp}(T_{KB}) = HB_P$, 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, $\mathit{lfp}(T_{KB}) = \bigcup_{i=1}^{n} T^i_{KB}(\emptyset) \;=\; T^n_{KB}(\emptyset)$ for some $n \geqslant 0$.*

Example 5.5 Suppose that the program P in $KB = (L, P)$ consists of the rules $r_1\colon b \leftarrow DL[S \uplus p; C](a)$ and $r_2\colon p(a) \leftarrow$, and L is the axiom $S \sqsubseteq C$. Then, KB is positive, and $\mathit{lfp}(T_{KB}) = \{p(a), b\}$, 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 $i \geq 0$.

Theorem 5.6 *Suppose $KB = (L, P)$ has a stratification λ 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*

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

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

Notice that $M_0 = \mathit{lfp}(T_{KB_0})$ 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 also the rule $r_3\colon q(x) \leftarrow not\ \neg b, not\ DL[S](x)$ is in P of Example 5.5. Then, the mapping λ 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 $M_0 = \mathit{lfp}(T_{KB_0}) = \{p(a), b\}$ 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 globally new atom $d_a(\mathbf{t})$.

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

$$d_a(\mathbf{c}) \leftarrow not\ \neg d_a(\mathbf{c}) \quad \text{and} \quad \neg d_a(\mathbf{c}) \leftarrow not\ d_a(\mathbf{c}) \tag{3}$$

for each dl-atom $a(\mathbf{c}) \in DL_P$. Intuitively, they "guess" the truth values of the dl-atoms of P.[1] 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 description logic 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 $I|HB_P$ (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.

[1] The guessing rules in (3) 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 [66].

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 $I^\star \subseteq HB_{P_{guess}}$ 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 [79].

Example 5.9 The following short 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:

$$nman(X) \leftarrow not\, pman(X);$$
$$pman(X) \leftarrow d_1(X);$$
$$d_1(lee) \leftarrow not\, \neg d_1(lee);$$
$$\neg d_1(lee) \leftarrow not\, d_1(lee);$$

This rewritten program has 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.

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 description logic 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) [85] and the extended closed-world assumption (ECWA) [40, 41], and of default reasoning, including Poole's [84] and Reiter's approach [86], can be implemented on top of a description logic knowledge base. Indeed, dl-programs are particularly well-suited for capturing 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 [78] can be emulated using dl-programs in a faithful way.

6.1 Closed-World Reasoning

Reiter's well-known closed-world assumption (CWA) [85][2] is acknowledged as an important reasoning principle for inferring negative information from a logical knowledge base 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.

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

Description logic 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.

Nonetheless, it is acknowledged that many Semantic Web application scenarios can not renounce to some form of closed-world reasoning [2, 47, 83]. Furthermore, the use of description logics for more expressive data models than the plain relational model, which has been proposed, e.g., in [16, 8] and advocated for enterprise application integration and data integration [65], also increases the interest in dealing with the CWA in this context.

Using dl-programs, the CWA may be easily expressed on top of an external KB which can be queried through suitable dl-atoms. We show this here for a description logic knowledge base L.

Intuitively, given a concept C, the ground atoms of C which are free for negation are determined by the following rule, where $\bar{c}$ is a designated predicate:

$$\bar{c}(X) \leftarrow not\ DL[C](X)\,.$$

For example, given

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

the CWA infers $\overline{man}(lee)$. We can set up similar rules

$$\bar{r}(X,Y) \leftarrow not\ DL[R](X,Y)$$

for a role R determining all ground atoms free for negation in the designated predicate $\bar{r}$. 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[3] the rules above, plus the rule

$$q(\mathbf{t}) \leftarrow DL[\lambda; Q](\mathbf{t}), \tag{4}$$

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 $Q(\vec{t})$ 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 $X = lee$.

As well-known, the CWA can lead to inconsistency. If in the above example, L contains the further axiom

$$person = man \sqcup woman,$$

then both $man(lee)$ and $woman(lee)$ are free for negation, since $L \not\models man(lee)$ and $L \not\models woman(lee)$, and thus $CWA(L)$ is unsatisfiable. We can easily check inconsistency of the CWA with a further rule

$$incons \leftarrow DL[\lambda; \bot](b),$$

[3] An infinite number of concepts can be defined starting from atomic concepts. We only consider those relevant for CWA which are explicitly addressed; otherwise, CWA leads to an infinite result. In line of this, we can easily accommodate a partial CWA (cf. [42]), in which only atoms over selected concepts and roles are free for negation.

where $\bot$ is the empty concept (entailment of $\bot(b)$, for an arbitrary constant b, is tantamount to inconsistency).

The problem with inconsistency in the CWA as presented above stems from disjunctive knowledge. 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., [74, 14] for an overview).

One of the most advanced refinements is the *extended closed-world assumption* (ECWA) [40, 41], which is intimately related to circumscription [68]. As in [40, 41], we adopt the following assumptions, which are common in a database context:

- the *domain-closure assumption* (DCA): $\forall x(x = c_1 \vee x = c_2 \vee \cdots x = c_n)$, which states that there are no individuals other than the individuals $c_1, \ldots, c_n$ which 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.

These assumptions are fulfilled by the semantics of dl-programs, and can also be expressed in $\mathcal{SHOIN}(\mathrm{D})$.

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 $p(\mathbf{c})$ can not 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 $\mathcal{P}$.

Syntactically, the ECWA is expressed by declaring all closed formulas α *free for negation in* T such that α contains no predicate from $\mathcal{Z}$ and that there is no disjunction $\gamma = \ell_1 \vee \cdots \vee \ell_m$ where each ℓ_i is either a positive ground atom with predicate from $\mathcal{P}$, or a ground literal with predicate from $\mathcal{Q}$, such that $T \vdash \alpha \vee \gamma$ and $T \nvdash \gamma$. Then, the ECWA of T with respect to $\langle \mathcal{P}, \mathcal{Z} \rangle$ is given by $ECWA(T; \mathcal{P}, \mathcal{Z}) = T \cup \{\neg\alpha \mid \alpha \text{ is free for negation in } T \text{ w.r.t. } \langle \mathcal{P}; \mathcal{Z} \rangle\}$.

Semantically, $ECWA(T; \mathcal{P}, \mathcal{Z})$ is characterized in terms of minimal models defined as follows. Given two interpretations M and N for T, we write $M \leq_{\mathcal{P},\mathcal{Z}} N$ 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 $N \leq_{\mathcal{P},\mathcal{Z}} M$ and $M \not\leq_{\mathcal{P},\mathcal{Z}} N$.

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 models of $ECWA(T; \mathcal{P}, \mathcal{Z})$ correspond to the minimal Herbrand models of T.

Proposition 6.1 ([40, 41]) *An interpretation M is a model of $ECWA(T; \mathcal{P}, \mathcal{Z})$ iff M is a $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal model of T.*

This result implies that $ECWA(T; \mathcal{P}, \mathcal{Z})$ is consistent when T is consistent. It also shows that $ECWA(T; \mathcal{P}, \mathcal{Z})$ semantically coincides with the parallel circumscription $CIRC(T; \mathcal{P}, \mathcal{Z})$ of the predicates from $\mathcal{P}$ in T while the predicates in $\mathcal{Z}$ are allowed to vary, which singles out precisely the $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal models of T [68].

Since we can view a description logic knowledge base L as a first-order theory with unary and binary predicates for concepts and roles, respectively, we can readily apply the ECWA to it. If we minimize in our

example knowledge base

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

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 can not 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 can not 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.

The above scheme can be easily formulated for encoding $\langle \mathcal{P}, \emptyset \rangle$-minimal models; assuming that L is consistent, we set up rules for each predicate p in $\mathcal{P}$ as above, and set λ accordingly. We next describe an encoding of the $\langle \mathcal{P}, \mathcal{Z} \rangle$-minimal models for an arbitrary L, where the predicates in $\mathcal{Q}$ and $\mathcal{Z}$ are encoded by different rules.

Definition 6.2 Let L be a description logic 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 $KB^{\text{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle} = (L, P)$ is built by constructing $\mathcal{P}$ as follows:

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

$$\overline{p}(\vec{X}) \leftarrow not\ p^+(\vec{X}), \tag{5}$$
$$p^+(\vec{X}) \leftarrow DL[\lambda; p](\vec{X}), \tag{6}$$

   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

$$\overline{p}(\vec{X}) \leftarrow not\ p^+(\vec{X}), \tag{7}$$
$$p^+(\vec{X}) \leftarrow not\ \overline{p}(\vec{X}). \tag{8}$$

3. $\mathcal{P}$ contains the rule

$$fail \leftarrow DL[\lambda'; \bot](b), not\ fail, \tag{9}$$

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

In this program, the rules (5) and (6) determine, similar as in the example above, 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 (6), but not those on $\mathcal{Z}$ since they are not relevant. The rules (7) and (8) simply guess the extension of the concepts and roles in $\mathcal{P}$ and $\mathcal{Q}$. The compatibility of the interpretation of all ground atoms is then checked with the rule (9); note that, by the minimality of the part on $\mathcal{P}$, no positive assumptions about $\mathcal{P}$ have to be fed to L (i.e., $p \uplus p^+$ is not needed in λ').

Theorem 6.3 *Let L be a description logic 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 $p^+(\mathbf{c}) \in M$.*

An immediate consequence of this result is that we can reduce query answering from a description logic 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 6.4 *Let L be a description logic 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 $Q(\mathbf{c})$, $L \models Q(\mathbf{c})$ 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, i.e., $dl_Q(\mathbf{c})$ belongs to every strong answer set of $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$.*

We remark that as for query answering, we may also modify the program $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$ by omitting all rules related to predicates in $\mathcal{Z}$, and by using the rule (4) encoding the query for CWA from above (in particular, if the concept resp. role of $\mathcal{Q}$ is from $\mathcal{Z}$), and ask for the cautious consequences $q(\mathbf{c})$ under strong answer set semantics.

Furthermore, we may add $not\ DL[p](\vec{X})$ in the body of the rule (5) without changing the answer sets, and similarly $not\ DL[p](\vec{X})$ and $not\ DL[\neg p](\vec{X})$ in the body of rule (7) and rule (8), respectively. These additions just prune assumptions made on p which are obviously not consistent with L itself.

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 help to minimize this set of facts.

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, description logic 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, difficult and cumbersome. However, overriding a "default" value is often a natural way for defining a subclass.

Example 6.5 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.

$$\begin{aligned} L = \{&redWine \sqsubseteq \neg whiteWine,\\ &lambrusco \sqsubseteq sparklingWine \sqcap redWine,\\ &sparklingWine(veuveCliquot),\ \ lambrusco(lambrusco\_di\_Modena)\}, \end{aligned}$$

We know that sparkling wine is usually white; however, we can not add the axiom $sparklingWine \sqsubseteq whiteWine$ to L without destroying L's consistency, since $Lambrusco$ is an exception. From L alone, we can not 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 [61].

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:

$$\begin{aligned} white(W) &\leftarrow DL[sparklingWine](W), not\ \neg white(W);\\ \neg white(W) &\leftarrow DL[whiteWine \uplus white; \neg whiteWine](W). \end{aligned}$$

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 [84] and Reiter's classical default logic [86] can be realized on description logics 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. Intuitively, we can view $g \in H$ as the default $\frac{\top : Mg}{g}$ in Reiter's logic, where $\top$ is any tautology.

We go here one step further and equip g with a precondition $pc(g)$, which is another (possibly) open formula which is instantiated along with g. It enables the instance g' of g if the instance $pc(g)'$ of $pc(g)$ is provable from L; roughly, this corresponds to the Reiter default $\frac{pc(g) : Mg}{g}$ (with some proviso, discussed below). Following [84], a *scenario* is a satisfiable set $\mathcal{F} \cup D$ where D consists of ground instances g' such

that $g \in H$ and $T \vdash pc(g)'$, and an *extension* of T is the set of consequences $Cn(T \cup S)$ of a maximal (with respect to $\subseteq$) scenario $T \cup S$.

In what follows, we assume that both $pc(g)$ and g are atoms $a(\vec{X})$ and $w(\vec{Y})$, respectively, where a and w are concepts or roles, and write the possible hypothesis as $a(\vec{X}) : w(\vec{Y})$. Intuitively, it maximizes the extension of w, whenever a is true in a description logic knowledge base L while maintaining consistency. In our example above, the possible hypothesis would be $sparklingWine(X) : whiteWine(X)$.

Given possible hypotheses $H = \{a_i(\vec{X}_i) : w_i(\vec{Y}_i), 1 \leq i \leq n\}$, we can model this behavior according to the above scheme as follows:

$$w_i^+(\vec{Y}_i) \leftarrow DL[a_i](\vec{X}_i), not\ \neg w_i^+(\vec{Y}_i), \tag{10}$$

$$\neg w_i^+(\vec{Y}_i) \leftarrow DL[\lambda; \neg w_i](\vec{Y}_i), \tag{11}$$

where p^+ is a predicate in the logic program for p, and $\lambda = w_1 \uplus w_1^+, \ldots, w_n \uplus w_n^+$, i.e., the update of L with instances of possible hypotheses to form a scenario. The answer set semantics effects that the update λ is maximal and, moreover, preserves consistency. If it would cause inconsistency, then $\neg w_i^+(X)$ would be derived for every $w_i(X)$, and hence no rule (10) 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 6.5 is of this form.

Formally, we have the following correspondence result.

Theorem 6.6 *Let L be a satisfiable description logic knowledge base, and let $H = \{a_i(\vec{X}_i) : w_i(\vec{Y}_i), 1 \leq i \leq n\}$ be a set of possible hypotheses. Furthermore, let $KB = (L, P)$ where P consists of all rules (10) and (11) for H, and define $scen(M) = L \cup \{w_i(\mathbf{c}) \mid w_i^+(\mathbf{c}) \in M\}$ for any interpretation M. Then:*

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

The following example shows a simple application of this result.

Example 6.7 Consider a circuit diagnosis scenario. We recall that in classical model-based diagnosis, one aims at finding diagnoses which maximize the set of working components. A Reiter diagnosis [87] of a system $S = (SD, COMP, OBS)$, where SD is the logical system description, $COMP$ the set of components, and OBS the set of observations, is a minimal (with respect to $\subseteq$) set $\Delta \subseteq COMP$ such that $SD \cup OBS \cup \{\neg ok(c) \mid c \in \Delta\} \cup \{ok(c) \mid c \in COMP \setminus \Delta\}$ is satisfiable; here, $ok(c)$ denotes that component c is working.

Assume that SD, $COMP$, and OBS are given by a description logic knowledge base L, which contains the concepts $comp$ and ok. Using the possible hypothesis $comp(X) : ok(X)$, the diagnoses of the system naturally correspond to the maximal scenarios $L \cup S$. Expressing the hypothesis by

$$working(X) \leftarrow DL[comp](X), not\ \neg working(X), \tag{12}$$

$$\neg working(X) \leftarrow DL[ok \uplus working; \neg ok](X), \tag{13}$$

the answer sets of $KB = (L, P)$ encode the diagnoses of S as described by Theorem 6.6. In particular, for component c, an answer set contains $working(c)$ iff c does not belong to the corresponding diagnosis.

By adding to the program in Theorem 6.6 further rules

$$p^+(\vec{X}) \leftarrow DL[\lambda; p](\vec{X}), \qquad \neg p^+(\vec{X}) \leftarrow DL[\lambda; \neg p](\vec{X}), \tag{14}$$

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 6.5, the rule

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

would export the red wines, and in particular $red(lambrusco\_di\_Modena)$ 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 (14), 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.

As mentioned above, a possible hypothesis $pc(g) : g$ roughly corresponds to the Reiter default $\frac{pc(g) \,:\, Mg}{g}$. The difference is that, in the definition above, the precondition $pc(g)$ refers to the original theory T, while in Reiter's default logic it refers to T augmented with all default conclusions. In particular, the above definition 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 6.8 Suppose that in the wine scenario, we have the further knowledge that white wine is usually served cold, expressed by $whiteWine(X){:}servedCold(X)$. From the program KB in Theorem 6.6, however, we can not conclude that Veuve Cliquot is served cold, since $servedCold^+(veuveCliquot)$ is not contained in the single strong answer set of KB. The reason is that the precondition of $whiteWine(veuveCliquot){:}servedCold(veuveCliquot)$, the fact $whiteWine(veuveCliquot)$, is not a consequence of L, but is only obtained after the application of $sparklingWine\,(veuveCliquot) : whiteWine(veuveCliquot)$.

In order to propagate conclusions from defaults to preconditions of defaults, we have to add these conclusions to the description logic knowledge base L when testing the preconditions. To this end, we modify rule (10) in the following way:

$$w_i^+(X) \leftarrow DL[\lambda; a_i](X), not\, \overline{w}_i(X). \tag{15}$$

Following this approach, we can in fact capture Reiter's default logic, which we show next.

6.2.2 Reiter's Default Logic

Recall that a default theory $T = \langle W, D\rangle$ consists of a set W of first-order sentences and a set D of defaults $\frac{\alpha \,:\, M\beta_1,\ldots,M\beta_n}{\gamma}$, where α, all β_i, and γ are (possibly open) first-order formulas. Reiter defines the extensions of a closed default theory T (i.e., where all defaults in T contain sentences only) as the fixpoints of the operator $\Gamma_T(S)$ as follows. For a set of sentences S, $\Gamma_T(S)$ is the least set of sentences such that

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

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

For a set D of defaults of the form $\frac{a(\vec{X}) : Mw(\vec{Y})}{w(\vec{Y})}$ (referred to as *normal defaults* in the literature), i.e., for possible hypotheses $a(\vec{X}) : w(\vec{Y})$ as in Section 6.2.1, over a description logic knowledge base L, we can express the extensions of $\langle L, D\rangle$ by the strong answer sets of a corresponding dl-program. Roughly, the latter program implements a guess-and-check strategy, according to which a sufficiently large part of an extension E, which includes all conclusions of applied defaults, is guessed using predicates $in\_w_i(\vec{Y_i})$ and $out\_w_i(\vec{Y_i})$ and described by an update $\lambda' = w_1 \uplus in\_w_1, \ldots, w_n \uplus in\_w_n$ of L. The candidate E is then checked using predicates $w_i^+(\vec{Y_i})$ to characterize $\Gamma_T(E)$, which is described by the update $\lambda = w_1 \uplus w_1^+, \ldots, w_n \uplus w_n^+$ of L.

Definition 6.9 Let L be a description logic knowledge base, and let $D = \{\delta_i = a_i(\vec{X_i}) : w_i(\vec{Y_i}), 1 \leq i \leq n\}$ be a set of defaults. 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 $w_i(\vec{Y})_i$ belongs to the extension E:

$$in\_w_i(\vec{Y_i}) \leftarrow not\ out\_w_i(\vec{Y_i}), \tag{16}$$
$$out\_w_i(\vec{Y_i}) \leftarrow not\ in\_w_i(\vec{Y_i}); \tag{17}$$

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

$$false \leftarrow DL[\lambda'; w_i](\vec{Y_i}), out\_w_i(\vec{Y_i}), not\ false, \tag{18}$$

   where $\lambda' = w_1 \uplus in\_w_1, \ldots, w_n \uplus in\_w_n$;

3. a rule for applying δ_i as in $\Gamma_T(E)$:

$$w_i^+(\vec{Y_i}) \leftarrow DL[\lambda; a_i](\vec{X_i}), not\ DL[\lambda'; \neg w_i](\vec{Y_i}), \tag{19}$$

   where $\lambda = w_1 \uplus w_1^+, \ldots, w_n \uplus w_n^+$;

4. rules which check whether E and $\Gamma_T(E)$ coincide:

$$false \leftarrow not\ DL[\lambda; w_i](\vec{Y_i}), in\_w_i(\vec{Y_i}), not\ false, \tag{20}$$
$$false \leftarrow DL[\lambda; w_i](\vec{Y_i}), out\_w_i(\vec{Y_i}), not\ false. \tag{21}$$

Example 6.10 Considering again our wine scenario with the two defaults $sparklingWine{:}white\text{-}\ Wine$ and $whiteWine{:}servedCold$, the program KB^{df} (which is not listed here for space reasons) has a single answer set M, which contains $white^+(veuveCliquot)$ and $servedcold^+(veuveCliquot)$. Thus, it is concluded that Veuve Cliquot is served cold. On the other hand, M neither contains $servedcold^+(lambrusco\_di\_Modena)$ nor $\neg servedcold^+(lambrusco\_di\_Modena)$, and nothing can be concluded about Lambrusco di Modena being served cold or not.

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

Theorem 6.11 *Let L be a description logic knowledge base, let $D = \{\delta_i = a_i(\vec{X}_i) : w_i(\vec{Y}_i),\ 1 \leq i \leq n\}$ be a set of defaults, and let $T = \langle L, D\rangle$. Then:*

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

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

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

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

is an extension of T.

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

By adding rules (14) 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 T, via brave and cautious reasoning from KB^{df}.

We note that the rules (18) can be eliminated from KB^{df}, at the price of introducing more dl-literals, by replacing $in\_w_i(\vec{Y}_i)$ in (20) with $DL[\lambda'; w_i](\vec{Y}_i)$ and $out\_w_i(\vec{Y}_i)$ in (21) with $not\ DL[\lambda; w_i](\vec{Y}_i)$. Furthermore, the guesses in Part 1 might be limited with techniques similar to those mentioned in the previous subsection.

Eventually, we remark that the above encoding can be extended to more general classes of defaults $\frac{\alpha : M\beta_1, \ldots, M\beta_n}{\gamma}$. In particular, this is straightforward, yet notationally cumbersome, for defaults where α and γ are conjunctions of literals and each β_i is a literal. The investigation of further fragments remains for future work.

6.3 DL-Safe Rules

DL-safe rules [78] represent one of the first attempts to couple rules with ontologies while keeping a full first-order semantics together with decidability. In order to ensure this, only a limited form of rules is allowed.

Intuitively, a *DL-safe program* is a description logic 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 6.12 Suppose L is a $\mathcal{SHOIN}(\mathrm{D})$ knowledge base, where $\mathbf{A}$, $\mathbf{R}_A$, and $\mathbf{R}_D$ are the atomic concept names, abstract role names, 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

$$h_1(\vec{Y}_1) \vee \cdots \vee h_m(\vec{Y}_m) \leftarrow b_1(\vec{X}_1), \ldots, b_n(\vec{X}_n), \tag{22}$$

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,[4] 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*

[4] Concept names from $\mathbf{A}$ are unary predicates, while role names from $\mathbf{R}_A \cup \mathbf{R}_D$ are binary predicates.

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

$$p(X) \leftarrow knows(X, Y), allDaugthers(Y), knows(X, Z), parent(Y, Z) \tag{23}$$

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 6.14 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 in the following way:

Definition 6.15 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

$$p^+(\vec{X}) \leftarrow not\, p^-(\vec{X}), \tag{24}$$
$$p^-(\vec{X}) \leftarrow not\, p^+(\vec{X}), \tag{25}$$

   where p^+ and p^- are new predicates;[5]

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

$$fail \leftarrow not\, h_1^+(\vec{Y}_1), \ldots, not\, h_m^+(\vec{Y}_m),\ b_1^+(\vec{X}_1), \ldots, b_n^+(\vec{X}_n), not\, fail; \tag{26}$$

   and

3. the rule

$$fail \leftarrow DL[\lambda; \bot](b), not\, fail, \tag{27}$$

   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.

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

Intuitively, rules (24) and (25) guess the extension of each predicate in P. Rule (26) checks satisfaction of the rule r, and rule (27) 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 $P\downarrow$,[6] is sufficient to capture the models of a combined knowledge base (L, P) with respect to ground atoms.

Lemma 6.16 *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 6.17 *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

$$q(\mathbf{t}) \leftarrow \chi(Q; \mathbf{t}), \tag{28}$$

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 $KB^{dls}_{Q(\mathbf{t})}$ the resulting dl-program. From Theorem 6.17, the following result is then easily obtained.

Corollary 6.18 *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 (27) can be dropped from $KB^{dls}_{Q(\mathbf{t})}$. Furthermore, equality in the DL-knowledge base and rules ($\approx$) can be emulated by treating $\approx$ like a role occurring in P, and adding further constraints which enforce that equal objects behave equally (i.e, $fail \leftarrow p(\vec{X}), not\ p(\vec{Y}), X_1 \approx Y_1, \ldots, X_n \approx Y_n, not\ fail$, for all predicates p in $\mathbf{P}$ where $\vec{X} = X_1 \ldots, X_n$ and $\vec{Y} = Y_1, \ldots . Y_n$). Finally, we remark that we can extend the program KB^{dls} to inductively built concepts, and similar for query answering.

[6] 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.

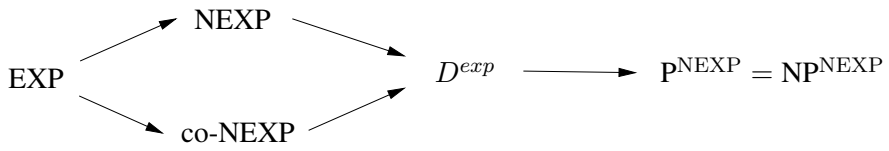

Figure 1: Containment between complexity classes.

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 [63, 64, 82]. 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 $\mathrm{P}^{\mathrm{NEXP}}$ 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 $\mathrm{NP}^{\mathrm{NEXP}}$ [48] 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.

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

Table I: 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

whether a knowledge base L in $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$) is satisfiable is complete for EXP [95, 55] (resp., NEXP, assuming unary number encoding; see [55] and the NEXP-hardness proof for $\mathcal{ALCQI}$ in [95], 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 I 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 non-monotonic reasoning in the propositional case [44, 24]; we leave a detailed study of the complexity of fragments of dl-programs for further work.

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.*

Table II: 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 III: 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

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 II and III, respectively, compactly summarize our complexity results for these problems for L from $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. Roughly, 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 I).

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 "$\neg$"-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 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. The reasons for this decision were mainly constrained human resources but also the fact that these existing engines have been professionally developed and are supposedly highly efficient.

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, programs are often structured in two hierarchic layers: a first, stratified layer at the bottom performs some preprocessing on the input data and a second, unstratified layer usually is aimed at encoding some nondeterministic choice and verification. It is therefore 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 [70]. For simplicity, our approach is to split the program only in two parts, having a fast routine for finding the answer set(s) of the lower layer, while the remaining subprogram will be solved by the guess-and-check method (if such a subprogram exists at all).

8.1 Splitting the Input Program

Lifschitz and Turner [70] 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 [70] 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) non-monotonic dl-atom.

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

For example, for $r : p(X) \leftarrow q(X), r(X)$, according to ($P_1$), we have the dependencies $p(X) \rightarrow_p q(X)$ and $p(X) \rightarrow_p r(X)$. Furthermore, for $r_1 : p(X) \leftarrow q(X), r(X)$ and $r_2 : q(Y) \leftarrow s(Y)$, condition ($P_2$) yields $q(X) \rightarrow_p q(Y)$. On the other hand, for $r_1 : p(X) \leftarrow DL[Student \uplus s; Person](X)$ and $r_2 : s(X) \leftarrow enrolled(X)$, in view of ($P_3$), $DL[Student \uplus s; Person](X) \rightarrow_p s(X)$ holds. As for negative dependencies, rule $r : flies(X) \leftarrow bird(X), not\ penguin(X)$ entails $flies(X) \rightarrow_n penguin(X)$ in view of (N_1). Finally, for rule $r : part(X) \leftarrow DL[P \uplus known; P](X)$, we get $part(X) \rightarrow_n DL[P \uplus known; P](X)$ according to condition (N_2).

We now define splitting sets as follows.

Definition 8.2 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 [70]. We consider two sets U, X of 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 program consisting of all such rules r' is denoted by $e_U(P, X)$.

Definition 8.3 Let U be a splitting set for a program $KB = (L, P)$. We call a pair $\langle X, Y \rangle$ of sets of literals a *solution* to KB with respect to U, if

- X is an answer set for $b_U(P)$,
- Y is an 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 8.4 *Let* U *be a splitting set for a dl-program* $KB = (L, P)$. *Then,* A *is an 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 8.5 *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 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 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 8.5 a *stratification splitting set* for a dl-program KB. The following property is an immediate consequence of Theorem 8.5.

Corollary 8.6 *Each dl-program has exactly one stratification splitting set.*

Example 8.7 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:

$paper(p_1)$;
$kw(p_1, Semantic\_Web)$;
$paper(p_2)$;
$kw(p_2, Bioinformatics)$;
$kw(p_2, Answer\_Set\_Programming)$;
$kw(P, K_2) \leftarrow kw(P, K_1),\ DL[hasMember](S, K_1), DL[hasMember](S, K_2)$;
$paperArea(P, A) \leftarrow DL[keywords \uplus kw; inArea](P, A)$;
$cand(X, P) \leftarrow paperArea(P, A),\ DL[Referee](X),\ DL[expert](X, A)$.

This program is positive, and thus can only have a single 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 8.4 that the answer sets of a dl-program $KB = (L, P)$ can be obtained by computing the unique answer set M of $b_U(P)$ (where U is the stratification splitting set) and then computing the 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 fixpoint (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].

8.2 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.

8.2.1 DL-Function Calls

One of the features of DL-reasoners which may be fruitfully exploited for speed up are non-ground queries. RACER 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. Given that the cost for accessing the DL-reasoner is high, in the case when several different ground instances $a(\mathbf{c}_1), a(\mathbf{c}_2), \ldots, a(\mathbf{c}_k)$ of the dl-atom $a(\mathbf{t})$ have be evaluated, it is a reasonable strategy to retrieve at once, using the apposite function call feature from the DL-reasoner, all instances of the concept C (resp., a role R) in $a(\mathbf{t}) = DL[\lambda; C](\mathbf{t})$. This allows to avoid issuing k separate calls for the single ground atoms $a(\mathbf{c}_1), \ldots, a(\mathbf{c}_k)$.

If the retrieval set has presumably many more than k elements, we can filter it with respect to $\mathbf{c}_1, \ldots, \mathbf{c}_k$, by pushing these instances to a DL-engine as follows. For the query concept C, we add in L axioms to the effect that $C'' = C \sqcap C'$, 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 yet more involved approximation method is introduced, given that $\mathcal{SHIF}(\mathrm{D})$ and $\mathcal{SHOIN}(\mathrm{D})$ 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.

8.2.2 DL-Caching

Whatever semantics is considered, a number of calls will be made to the DL-engine. Therefore, it is 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 some 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 $\langle J, o\rangle$ such that $J \subseteq I^\lambda$ if $v = true$, or $J \supseteq I^\lambda$ if $v = undefined$. If such a J exists, conclude that $I(a) = v$.

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

8.3 System Prototype

The architecture of our system prototype NLP-DL, which has been described in [29, 28], is depicted in Figure 2. The system comprises different modules, each of which is coded in the PHP scripting language; the overhead 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[7] 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 [66] was used as answer set solver and RACER [46] 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) [34], and (3) under the answer set semantics with preliminary computation of the WFS.

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

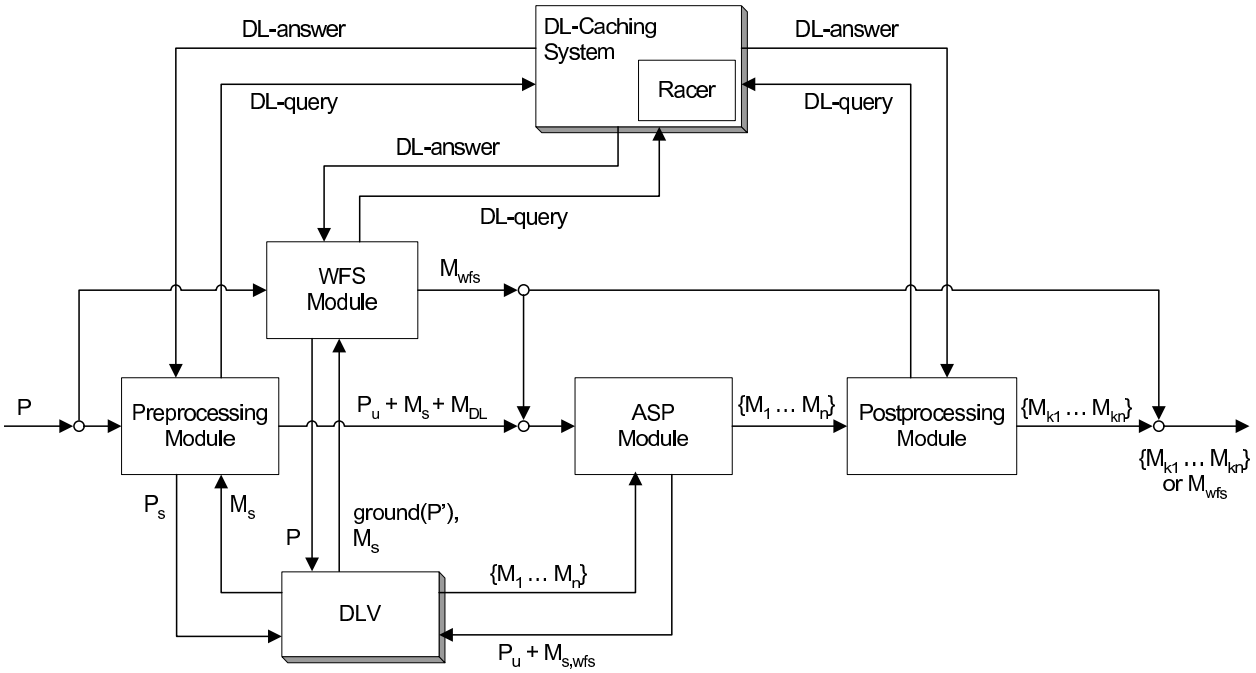

Figure 2: System architecture of the dl-program evaluation prototype

The preprocessing module evaluates all dl-atoms without any input (producing the set M_{DL}), applies the splitting method (separating the unstratified subprogram P_u), and computes the single answer set of the stratified subprogram (M_s). The ASP module implements the guessing part of the evaluation, using DLV for the answer set generation ($\{M_1 \dots M_n\}$). This result is streamed to a post-processing module, which carries out the verification of each incoming answer set according to the weak resp. strong answer set semantics, returning the final result $\{M_{k1} \dots M_{kn}\}$. The WFS module is used for computing the well-founded semantics M_{wfs} of the knowledge base [34], which conservatively extends the canonical well-founded semantics of logic programs [97] to dl-programs, retaining many of its attractive properties. In particular, it approximates the intersection of all strong answer sets, and thus preliminary computation of the WFS can be exploited to reduce guesses for the generation of answer sets [29].

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, 81], and for discussions of general issues that come up when combining rules and ontologies to [18, 27, 91].

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

9.1 Interaction of Rules and Ontologies with Strict Semantic Separation

In this setting, a (usually nonmonotonic) language plays a central role in the Rules Layer, while OWL/RDF flavors keep their purpose of description languages, not aimed at intensive reasoning jobs, in the underlying Ontology Layer. The two layers are kept strictly separate and 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 description logic 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, 71, 72, 73, 99, 102]. 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 [71, 72] extend dl-programs by probabilistic uncertainty, and similarly fuzzy dl-programs [73] by fuzzy vagueness. An extension of dl-programs to handle priorities is conceived in [102]. In [99], 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 [93] 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 [45] fragment marks one end of this spectrum while the undecidable SWRL [58] 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. [45] 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 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 [67], and Rosati [88, 89] 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 $\mathcal{ALC}$. 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 [67] presented a combination of Horn rules with the description logic $\mathcal{ALCNR}$, 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. [78] adopted DL-safety like Donini et al., but permitted both concepts and roles as constraints freely in the heads and bodies of rules. They showed decidability of the combination with the more expressive description logic $\mathcal{SHIQ}$ (cf. Section 6.3), thus bringing us (close to) a decidable extension of OWL with rules. Horrocks et al.'s SWRL [58], instead, which extends OWL by rules that violate the DL-safety restriction, is undecidable. Another approach [50] in the direction of Motik et al. shows decidability for query answering in $\mathcal{ALCHO}$-$\mathcal{Q}(\sqcup, \sqcap)$ 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 [88, 89] combined disjunctive datalog (with classical and default negation) with $\mathcal{ALC}$ based on a generalized answer set semantics. Like Levy and Rousset [67], he allowed besides concepts also roles as constraints in rule bodies, and, similar to Donini et al. [21], DL-safety was not 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 [92, 91], which builds on his previous work [88, 89], 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 a unnegated atom in the body of that rule; this is because $\mathcal{DL}{+}log$ uses an infinite domain under 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 [78], 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. For example, given the simple person knowledge base L from Section 6.1 and the two dl-rules $p(X) \leftarrow DL[man](X)$ and $p(X) \leftarrow DL[woman](X)$,

we can not conclude $p(lee)$ since neither $man(lee)$ nor $woman(lee)$ is a consequence of L; from the corresponding $\mathcal{DL}{+}log$ program, $p(lee)$ is concluded since in each model either $man(lee)$ or $woman(lee)$ is true, and thus also $p(lee)$. We remark, however, that by using auxiliary concepts or more expressive queries to the DL knowledge base [26], the behavior of dl-programs in this respect can be mitigated.

- The coupling, as realized in dl-programs, aims at facilitating interoperability of existing reasoning systems and software, such as DLV and RACER. For this, a combination of the underlying formalisms at the extensional level is needed. On the other hand, the loose coupling requires a bridging between the two worlds of ontologies and rules, which has to be provided by the user. In particular, this applies to the individuals at the instance level.

- The concept of dl-atom makes straightforward the introduction of extensions to dl-programs to integrate ontologies even in different formats; 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.

The most recent work of Motik and Rosati [75, 76] 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 [69]. 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 [76], 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, [76] 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 can not be captured using MKNF rules.

Other recent works which aim at combining rules and ontologies through uniform first-order nonmonotonic formalisms are [19, 20]. 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. [96], Alsaç and Baral [1, 7], Swift [94], Hufstadt et al. [62], and Heymans and Vermeir [51, 52]. In more detail, Van Belleghem et al. [96] analyze the close relationship between description logics and open logic programs, and present a mapping of description logic 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 [94] 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 Hufstadt et al. [62] 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 [77].

Finally, Heymans and Vermeir [51, 52] 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 non-monotonic 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 [66] and RACER [46], 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 [78] 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 [99] 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 $\mathcal{SHIF}(\mathrm{D})$ or $\mathcal{SHOIN}(\mathrm{D})$, 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 [16, 8], the interest in CQs and UCQs on DL knowledge bases is increasing [15, 43, 80], 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 description logic 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 [66]. Other extensions concern different semantics of the rules; in [34], a well-founded semantics for dl-programs has been defined, and in [100], a semantics based on defeasible logic. Other extensions concern the consideration of probabilities [71, 72], fuzziness [73], and of rule priorities [102].

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.

Acknowledgments

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 the KR 2004 preliminary version of this paper [33], whose constructive comments helped to improve our work.

A Appendix: Further Details on the Reviewer Selection Example

The description logic knowledge base L_S of the dl-program $KB_S = (L_S, P_S)$ that is described in Example 4.1 is partially given below (note that in our current prototype implementation based on RACER, the description logic knowledge base L_S as well as the logic program P_S have to be extended by workarounds, since RACER does not support individuals as part of concept expressions):

$$\geq 1\ \mathit{title} \sqsubseteq \mathit{Publication}; \qquad \top \sqsubseteq \forall \mathit{title}.\mathit{string};$$
$$\geq 1\ \mathit{year} \sqsubseteq \mathit{Publication}; \qquad \top \sqsubseteq \forall \mathit{year}.\mathbb{N};$$
$$\geq 1\ \mathit{firstname} \sqsubseteq \mathit{Person}; \qquad \top \sqsubseteq \forall \mathit{firstname}.\mathit{string};$$
$$\geq 1\ \mathit{lastname} \sqsubseteq \mathit{Person}; \qquad \top \sqsubseteq \forall \mathit{lastname}.\mathit{string};$$
$$\geq 1\ \mathit{keyword} \sqsubseteq \mathit{Paper}; \qquad \top \sqsubseteq \forall \mathit{keyword}.\mathit{Kw};$$
$$\geq 1\ \mathit{cites} \sqsubseteq \mathit{Paper}; \qquad \top \sqsubseteq \forall \mathit{cites}.\mathit{Paper};$$
$$\geq 1\ \mathit{contains} \sqsubseteq \mathit{Area}; \qquad \top \sqsubseteq \forall \mathit{contains}.\mathit{Kw};$$
$$\geq 1\ \mathit{hasAuthor} \sqsubseteq \mathit{Paper}; \qquad \top \sqsubseteq \forall \mathit{hasAuthor}.\mathit{Person};$$
$$\geq 1\ \mathit{expert} \sqsubseteq \mathit{Person}; \qquad \top \sqsubseteq \forall \mathit{expert}.\mathit{Area};$$
$$\geq 1\ \mathit{inArea} \sqsubseteq \mathit{Paper}; \qquad \top \sqsubseteq \forall \mathit{inArea}.\mathit{Area};$$
$$\geq 1\ \mathit{hasMember} \sqsubseteq \mathit{TopicCluster}; \qquad \top \sqsubseteq \forall \mathit{hasMember}.\mathit{Kw};$$

$$\mathit{isContainedIn} = \mathit{contains}^{-};$$
$$\mathit{isAuthorOf} = \mathit{hasAuthor}^{-};$$
$$\mathit{isMemberOf} = \mathit{hasMember}^{-};$$

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

$$\exists \mathit{inArea}.\{A\} = \exists \mathit{keyword}.(\exists \mathit{isContainedIn}.\{A\});$$
$$\exists \mathit{expert}.\{\mathrm{c}\} = \exists \mathit{isAuthorOf}.(\exists \mathit{inArea}.\{\mathrm{c}\}), \quad \mathrm{c} \in \{A, B, C, D, E\};$$

$$\mathit{Kw}(\mathit{Belief\_Revision}); \quad \mathit{Kw}(\mathit{Frame\_Systems});$$
$$\mathit{Kw}(\mathit{Intelligent\_Agents}); \quad \mathit{Kw}(\mathit{Bioinformatics});$$
$$\ldots$$
$$\mathit{Area}(A); \quad \mathit{contains}(A, \mathit{Belief\_Revision}); \quad \mathit{contains}(A, \mathit{Default\_Reasoning});$$
$$\mathit{Area}(B); \quad \mathit{contains}(B, \mathit{Frame\_Systems}); \quad \mathit{contains}(B, \mathit{Ontologies});$$
$$\mathit{Area}(C); \quad \mathit{contains}(C, \mathit{Semantic\_Web});$$
$$\mathit{Area}(D);$$
$$\ldots$$
$$\mathit{TopicCluster}(T_1); \quad \mathit{hasMember}(T_1, \mathit{Semantic\_Web}); \quad \mathit{hasMember}(T_1, \mathit{OWL});$$
$$\mathit{hasMember}(T_1, \mathit{Ontologies});$$
$$\mathit{TopicCluster}(T_2); \quad \mathit{hasMember}(T_2, \mathit{Coherence\_and\_Coordination});$$
$$\ldots$$
$$\mathit{Person}(\mathit{per}_1); \quad \mathit{firstname}(\mathit{per}_1, \text{“}\mathit{Vladimir}\text{”}); \quad \mathit{lastname}(\mathit{per}_1, \text{“}\mathit{Lifschitz}\text{”});$$
$$\mathit{Person}(\mathit{per}_2); \quad \mathit{firstname}(\mathit{per}_2, \text{“}\mathit{Michael}\text{”}); \quad \mathit{lastname}(\mathit{per}_2, \text{“}\mathit{Gelfond}\text{”});$$
$$\ldots$$
$$\mathit{Referee}(\mathit{per}_1);$$

$Referee(per_2)$;
...
$Paper(pub_1)$;
$title(pub_1,$ "*Classical Negation in Logic Programs and Disjunctive Databases*");
$year(pub_1,$ "*1991*");
$hasAuthor(pub_1, per_1)$; $hasAuthor(pub_1, per_2)$;
$keyword(pub_1, Answer\_Set\_Programming)$;
$keyword(pub_1, Disjunctive\_Logic\_Programming)$.
...

In addition to the dl-rules (1)–(9), the logic program P_S of the dl-program $KB_S = (L_S, P_S)$ of Example 4.1 also contains the following facts:

$author(per_1)$; $author(per_2)$; $author(per_3)$; ...
$area(A)$; $area(B)$; $area(C)$; $area(D)$;
$cluster(T1)$; $cluster(T2)$;
$key(Belief\_Revision)$;
$key(Nonmonotonic\_Reasoning)$; $key(Answer\_Set\_Programming)$; ...

B Appendix: Proofs for Section 4

Proof of Lemma 4.2. Suppose that $I_1, I_2 \subseteq HB_P$ are both 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 4.5. Let $\lambda : HB_P \cup DL_P \to \{0,1,\ldots,k\}$ 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 $KB_i = (L_i, P_i)$ such that $M_i|HB^\star_{P_{i-1}} = M_{i-1}|HB^\star_{P_{i-1}}$. 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 $J|HB^\star_{P_i} \neq J|HB^\star_{P_i}$. Let j be a minimal such i. Then, J is a model of KB_j. Furthermore, if $j > 0$, then $J|HB^\star_{P_{j-1}} = J|HB^\star_{P_{j-1}}$. But this contradicts M_j being the least model of KB_j such that $M_j|HB^\star_{P_{j-1}} = M_{j-1}|HB^\star_{P_{j-1}}$. This shows that M_k is a minimal model of KB. □

Proof of Theorem 4.8. 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 4.9. (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 a 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, 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 4.10. 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 4.9, KB has no strong answer set.
Now assume that KB is stratified. Let λ 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,

- $I|HB^\star_{P_0}$ is the least among all models $J \subseteq HB^\star_{P_0}$ of $(L, sP_0{}_L^I)$, and
- if $i > 0$, then $I|HB^\star_{P_i}$ is the least among all models $J \subseteq HB^\star_{P_i}$ of $(L, sP_i{}_L^I)$ with $J|HB^\star_{P_{i-1}} = I|HB^\star_{P_{i-1}}$.

It thus follows that

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

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 4.14. 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 4.15. 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 $I \models_L H(r') = H(r)$. This shows that $I \models_L r$. Hence, I is a model of KB. □

Proof of Theorem 4.16. Immediate by the observation that $wP_L^I = (P_L^I)^I$. □

Proof of Theorem 4.17. 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, there exists a model $J \subset I$ of wP_L^I. Hence, J is also a model of (L, sP_L^I). But this contradicts that I is the least model of (L, sP_L^I). This shows that I the least model of wP_L^I. That is, I is a weak answer set of KB. □

C Appendix: 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 $M_i | HB^\star_{P_{i-1}} = M_{i-1} | HB^\star_{P_{i-1}}$ 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 in (3) 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 $I^\star \subseteq HB_{P^\star} = HB_{P_{guess}}$ 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 4.16, 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 defined as in Theorem 4.16. 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 $I^\star \subseteq HB_{P_{guess}}$ of P_{guess} such that $d_a(\mathbf{c}) \in I^\star$ iff $I^\star \models_L a(\mathbf{c})$, for all $a(\mathbf{c}) \in DL_P$. □

D Appendix: 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 $KB^{\mathrm{ECWA}}_{\langle \mathcal{P}, \mathcal{Z} \rangle}$. 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 (9). 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, $\overline{p}(c) \in M$ implies $N \models \neg p(c)$. So, N must be a model of L such that $N \leq_{\mathcal{P},\mathcal{Z}} M'$. Assuming that $M' \not\leq_{\mathcal{P},\mathcal{Z}} N$, 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 (6) 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, $M' \leq_{\mathcal{P},\mathcal{Z}} N$ holds. Since $N \leq_{\mathcal{P},\mathcal{Z}} M'$ 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, rule (9) is eliminated in building the strong dl-transform sP_L^M of P relative to L and M. Next, for each ground atom $p(c)$ such that p is from $\mathcal{Q}\cup\mathcal{Z}$, by construction sP_L^M contains the fact $p^+(c)$ (emerging from rule (8)) iff $p^+(s) \in M$. Furthermore, for each ground atom $p(c)$, sP_L^M contains the fact $\overline{p}(c)$ (emerging from rule (5) or (7)) iff $\overline{p}(s) \in M$.

Consequently, M is 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 6.6. 1) Let M be a strong answer set of KB, and consider $scen(M) = L \cup \{w_i(\mathbf{c}) \mid w_i^+(\mathbf{c}) \in M\}$. First we show that $scen(M)$ is a scenario. By the rules of form (10), $w(\mathbf{c})$ can be in $scen(M)$ only if there is some instance $a_i(\mathbf{c}')$:$w_i(\mathbf{c})$ of a possible hypothesis in H such that $L \models a_i(\mathbf{c}')$. Furthermore, $scen(M)$ is satisfiable. Indeed, if this were not the case, then the rules of form (11) would include in M all literals $\neg w_i^+(\mathbf{c})$ where $w_i(\mathbf{c})$ is an instance of $w_i(\vec{Y}_i)$, for all $1 \leq i \leq n$. Hence, no rule instance of (10) is applicable, and thus M contains no positive literals $w_i^+(\mathbf{c})$. 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 atom $w_i(\mathbf{c}) \notin scen(M)$, and since S is satisfiable, we have that $scen(M) \not\models \neg w_i(\mathbf{c})$; that is, $L(M;\lambda) \not\models \neg w_i(\mathbf{c})$. Hence, $\neg w_i^+(\mathbf{c})$ is not in M, since it is not derivable by the rules of form (11). This means that the strong reduct sP_L^M contains a rule $w_i^+(\mathbf{c}) \leftarrow DL[a_i](\mathbf{c}')$ such that $M \models_L a_i(\mathbf{c}')$. Hence, $w_i^+(\mathbf{c}) \in M$, which means $w_i(\mathbf{c}) \in scen(M)$. This is a contradiction, which proves maximality of $scen(M)$.

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

$$M = \{w_i^+(\mathbf{c}) \mid w_i(\mathbf{c}) \in S\} \cup \{\neg w_i^+(\mathbf{c}) \mid S \models \neg w_i(\mathbf{c})\},$$

where $w_i(\mathbf{c})$ ranges over all instances of $w_i(\vec{Y}_i)$, $1 \leq i \leq n$. We show that M is a strong answer set of KB.

First, we show that M is a model of the strong reduct sP_L^M. By definition of M, clearly all rule instances of (11), which belong to sP_L^M, are satisfied. Furthermore, each rule in sP_L^M stemming from a rule of form

(10) is satisfied: if $M \models_L DL[a_i](\mathbf{c}')$ and $S \not\models \neg w_i(\mathbf{c})$, then, by maximality of $L \cup S$, $w_i(\mathbf{c}) \in S$, and thus $w_i^+(\mathbf{c}) \in M$ by construction. Finally, M is the least model of sP_L^M: any model $N \subseteq M$ contains $w_i^+(\mathbf{c})$ iff $w_i^+(\mathbf{c}) \in M$; since thus $L(N;\lambda) = L(M;\lambda)$, N also contains $\neg w_i^+(\mathbf{c})$ iff $\neg w_i^+(\mathbf{c}) \in M$. This proves that M is a strong answer set of KB. □

Proof of Theorem 6.11. For sets of formulas S and S', we denote by $conc(S, S')$ the set of all ground atoms $w_i(\mathbf{c})$ such that there is an instance $a_i(\mathbf{c}') : w_i(\mathbf{c})$ of a default from D such that $a_i(\mathbf{c}') \in S$ and $\neg w_i(\mathbf{c}) \notin S'$. Then, we recall that by Reiter's characterization of extensions in terms of generating defaults [86], $E = Cn(L \cup conc(E, E))$ holds for each extension of T.

1) Suppose $E = L(M;\lambda')$ is an extension of T. Define $M = \{in\_w_i(\mathbf{c}) \mid w_i(\mathbf{c}) \in E\} \cup \{out\_w_i(\mathbf{c}) \mid w_i(\mathbf{c}) \notin E\} \cup \{w_i^+(\mathbf{c}) \mid w_i(\mathbf{c}) \in conc(E, E)\}$. 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_L^M. It remains to show that M is the least model of sP_L^M. Let $N \subseteq M$ be the least model of sP_L^M. Clearly, N and M coincide on all predicates in_w_i and out_w_i, $i = 1, \ldots, n$. Let $S = Cn(L \cup \{w_i(\mathbf{c}) \mid w_i^+(\mathbf{c}) \in N\}) = Cn(L(N;\lambda))$ $(\subseteq E)$. Since N is a model of sP_L^M, the rules stemming from (19) imply that $\{w_i^+(\mathbf{c}) \mid w_i \in conc(S, E)\} \subseteq N$, and thus $conc(S, E) \subseteq S$; therefore, S satisfies conditions 1-3 of $\Gamma_T(E)$. By minimality of $\Gamma_T(E)$, it follows $S = \Gamma_T(E) = E$. Since $conc(S, E) = conc(E, E)$, it follows that $M \subseteq N$. Hence, $M = N$ is the least model of sP_L^M. 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 (16) and (17), M contains for each ground instance $w_i(\mathbf{c})$ of $w_i(\vec{Y}_i)$ exactly one of $in\_w_i(\mathbf{c})$ and $out\_w_i(\mathbf{c})$. Moreover, since $w_i \uplus in\_w_i$ occurs in λ', by the rules (18) $in\_w_i(\mathbf{c})$ belongs to M iff $L(M;\lambda') \models w_i(\mathbf{c})$ (equivalently, $w_i(\mathbf{c}) \in E$). Furthermore, by the rules (19), $Cn(L(M;\lambda)) = Cn(L(M;\lambda'))$ must hold.

Thus, it remains to show that $E = \Gamma_T(E)$. We first show that E satisfies conditions 1-3 of $\Gamma_T(E)$, which means $\Gamma_T(E) \subseteq E$. 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., $concl(E, E) \subseteq E$. Let $\delta_i' = a_i(\mathbf{c}') : w_i(\mathbf{c})$ be an instance of default δ_i such that $\neg w_i(\mathbf{c}) \notin E$ and $a_i(\mathbf{c_i}) \in E$. Since $L(M;\lambda) \equiv L(M;\lambda')$, we have $M \models_L DL[\lambda; a_i](\mathbf{c}')$ and $M \not\models_L DL[\lambda'; w_i](\mathbf{c})$. Hence, by the rules of form (19), M must contain $w_i^+(\mathbf{c})$, which implies that $w_i(\mathbf{c}) \in L(M;\lambda)$. Consequently, $w_i(\mathbf{c}) \in E$. This proves $concl(E, E) \subseteq E$. Thus E, satisfies conditions 1-3 of $\Gamma_T(E)$.

We finally show that $E \subseteq \Gamma_T(E)$. Let N result from M by removing each atom $w_i^+(\mathbf{c})$ such that $w_i(\mathbf{c}) \notin conc(\Gamma_T(E), E)$. Since $conc(\Gamma_T(E), E) \subseteq \Gamma_T(E)$, N is a model of the strong reduct sP_L^M. Since M is the least model of sP_L^M, $M = N$ follows, and $\Gamma_T(E)$ contains each $w_i(\mathbf{c})$ such that $w_i^+(\mathbf{c}) \in M$; hence, $L(M;\lambda) \subseteq \Gamma_T(E)$. Since $Cn(L(M;\lambda)) = Cn(L(M;\lambda')) = E$, it follows $E \subseteq \Gamma_T(E)$. □

Proof of Theorem 6.17. 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 (26), it follows that $\mathcal{I}$ satisfies $P\downarrow$. Hence, $\mathcal{I} \models (L, P\downarrow)$. Thus by Lemma 6.16, 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 $\mathcal{I}$ as described. We first show that M is a model of the strong reduct $P' = sP^{dls\,M}_{\;L}$. Clearly M satisfies each rule in P' which stems from (24) or (25). Next, since $\mathcal{I}$ satisfies each ground instance of every rule r in P. M satisfies each rule in P' which stems from (26). 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}. □

E Appendix: Proofs for Section 7

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 4.10, 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, we explore (one by one) the exponentially many possible inputs of the dl-atoms in $ground(P)$. For each input, evaluating the dl-atoms and removing them from $ground(P)$ is feasible in exponential time. Since we are then left with an ordinary stratified program KB', by Theorem 4.16, we try to compute $M_{KB'}$ by (a sequence of) fixpoint iterations, and check compliance with the inputs 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 description logic 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 4.10, 4.15, and 4.17 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 4.8 and 4.14, 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 $\tau\colon U(s,t) \to D$ 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*.

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.

To prove the NEXP-membership in the positive case, observe that a positive 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 included in I. A suitable I and S, 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, observe first that for each dl-program KB, the number of ground dl-atoms a is polynomial, and every ground dl-atom a has in general exponentially many different concrete inputs I_p (that is, interpretations I_p of its input predicates $p = p_1, \ldots, p_m$), but each of these concrete inputs I_p has a polynomial size. Furthermore, notice that during the computation of the canonical model of a positive dl-program by fixpoint iteration, any ground dl-atom a needs to be evaluated only polynomially often, as its input can increase only that many times.

We can thus guess inputs I_p for all dl-atoms, and evaluate them with a NEXP oracle in polynomial time. For the (monotonic) ones remaining in sP_L^I, we can further guess a chain $\emptyset = I_p^0 \subset I_p^1 \subset \cdots \subset I_p^k = I_p$, along which their inputs are increased in a fixpoint computation for sP_L^I, and evaluate the dl-atoms in it in polynomial time with a NEXP oracle. We then 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 description logic 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 [95] (which is based on a reduction from simple Turing machines to domino systems by Börger et al. [13]). 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".

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 Σ (consisting of 0, 1, and the blank symbol $''\ ''$). By Theorem 6.1.2 of [13], 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. Here, D is defined as the set of all triples of elements from $\Sigma \cup Q \times \Sigma \cup \{\#, e\}$, where Q is the set of all states of M', and $\#$ and e are two fresh symbols. Moreover, the linear-time reduction *trans* that transforms any string $b = b_0\, b_1 \ldots b_{n-1}$ over Σ into a string $w = w_0 w_1 \ldots w_{n-1}$ over D (of the same length) is defined as follows (where q_0 is the start state of M'):

$$\begin{array}{rcl} w_0 & = & (\#, (q_0, b_0), b_1), \\ w_1 & = & ((q_0, b_0), b_1, b_2), \\ w_2 & = & (b_1, b_2, b_3), \\ & \vdots & \\ w_{n-1} & = & (b_{n-2}, b_{n-1}, \text{`` ''}). \end{array}$$

As shown in [95], Lemma 5.18 and Corollary 5.22, for domino systems $\mathcal{D} = (D, H, V)$ and initial conditions $w = w_0 \ldots w_{n-1}$, there exist description logic knowledge bases L_n, $L_{\mathcal{D}}$, and L_w in $\mathcal{SHOIN}(\mathbf{D})$ (which can be constructed in polynomial time in n from $\mathcal{D}$ and w) 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. Informally, L_n encodes the torus $U(2^{n+1}, 2^{n+1})$, $L_{\mathcal{D}}$ represents the domino system $\mathcal{D}$, and L_w encodes the initial condition w. Intuitively, the elements of the torus $U(2^{n+1}, 2^{n+1})$ are encoded by objects, and any mapping $\tau\colon U(2^{n+1}, 2^{n+1}) \to D$ satisfying the compatibility constraints is encoded by the membership of these objects to concepts C_d with $d \in D$, while L_w explicitly represents some such memberships to encode the initial condition w. More precisely, L_w has the form $\{C_{i,0} \sqsubseteq C_{w_i} \mid i \in \{0, 1, \ldots, n-1\}\}$, where every $C_{i,0}$ is a concept containing exactly the object representing $(i, 0) \in U(2^{n+1}, 2^{n+1})$.

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

$$\begin{array}{rcl} L & = & L_n \cup L_{\mathcal{D}} \cup \{C_{i,0} \sqcap S_{i,d} \sqsubseteq C_d \mid i \in \{0, 1, \ldots, n-1\},\, d \in D\} \cup \\ & & \{C_{i,0}(o_i) \mid i \in \{0, 1, \ldots, n-1\}\}, \\ P & = & \{\neg b^l_{2l-2}(0) \leftarrow\} \cup \bigcup_{j=0}^{l} P^j, \end{array}$$

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 $j \in \{1, \ldots, l\}$:

$$\begin{array}{rcl} P^0_v & = & \{b^0_{-2i}(v_i) \leftarrow \mid i \in \{1, \ldots, k\}\} \cup \{b^0_{-2i+1}(1) \leftarrow \mid i \in \{1, \ldots, k\}\}, \\ P^j_v & = & \{b^j_{-i}(x) \leftarrow b^{j-1}_{-i}(x) \mid i \in \{1, \ldots, 2k\}\}. \end{array}$$

2. P^0_q initializes the rest of the input of the first oracle call with "dummy" bits, and every P^j_q 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:

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

3. Every $P^j_{w\leftarrow b}$ 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 $\mathcal{D}$. That is, $P^j_{w\leftarrow b}$ is a positive program consisting of $(n-2)\cdot 8 + 2\cdot 4$ ground rules, which are straightforward (and thus omitted here).

4. Every $P^j_{s\leftarrow w}$ 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 $s^j_{i,d}$:

$$P^j_{s\leftarrow w} \quad = \quad \{s^j_{i,d}(o_i) \leftarrow w^j_i(d) \mid i \in \{0,1,\ldots,n-1\},\ d \in D\}\,.$$

Observe then that M accepts v iff the last oracle call returns "yes". The latter is equivalent to $b^l_{2l-2}(1)$ being derived from KB and thus $b^l_{2l-2}(0)$ 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. □

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, 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 the two rules $p \leftarrow l$ and $\neg p \leftarrow l$ (resp., the two rules $p \leftarrow not\ l$ and $\neg p \leftarrow not\ l$), where p is a fresh propositional symbol. Adding these rules does 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.

The lower complexity bounds hold by a reduction from the complement of answer set existence (resp., answer set existence itself), since a dl-program $KB = (L, P)$ 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. Adding the rule $\neg p \leftarrow$ (resp., $p \leftarrow$) 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. Using the same line of argumentation as in the proof of Theorem 7.3, 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 the two rules $p \leftarrow l$ and $\neg p \leftarrow l$

(resp., the two rules $p \leftarrow not\ l$ and $\neg p \leftarrow not\ l$), where p is a fresh propositional symbol. In all cases except for brave reasoning from positive dl-programs, adding these 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 from the membership in $\mathrm{P}^{\mathrm{NEXP}}$ of deciding weak answer set existence in the stratified case, since (as argued above) a classical literal $l \in HB_P$ belongs to some weak answer set of the positive dl-program $KB = (L, P)$ iff the stratified dl-program $KB' = (L, P \cup \{p \leftarrow not\ l,\ \neg p \leftarrow not\ l\})$, where p is a fresh propositional symbol, has a weak answer set.

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 $l \notin I$. The latter is equivalent to: (i) 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 included in I, and (ii) there exists no interpretation I with $l \notin I$ and 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 included in 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}.

The lower complexity bounds for all above cases except for brave reasoning from positive dl-programs hold by a reduction from answer set non-existence (resp., existence), using the same line of argumentation as in the proof of Theorem 7.3, since a dl-program $KB = (L, P)$ 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. Adding the rule $\neg p \leftarrow$ (resp., $p \leftarrow$) 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 [95], 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. As shown in [95], Lemma 5.18 and Corollary 5.22, 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 description logic knowledge bases L_n, $L_{\mathcal{D}}$, L_v, and L_w in $\mathcal{SHOIN}(\mathbf{D})$ (which can be constructed in polynomial time in n from $\mathcal{D}$, v, and w) such that (a) $L_n \cup L_{\mathcal{D}} \cup L_v$ is satisfiable iff $\mathcal{D}$ tiles the torus $U(2^{n+1}, 2^{n+1})$ with initial condition v, and (b) $L_n \cup L_{\mathcal{D}} \cup L_w$ is unsatisfiable iff

$\mathcal{D}$ does not tile $U(2^{n+1}, 2^{n+1})$ with initial condition w. Informally, L_n encodes the torus $U(2^{n+1}, 2^{n+1})$, and $L_{\mathcal{D}}$ represents the domino system $\mathcal{D}$, while L_v and L_w encode the initial conditions v and w, respectively. Intuitively, the elements of the torus $U(2^{n+1}, 2^{n+1})$ are encoded by objects, and any mapping $\tau\colon U(2^{n+1}, 2^{n+1}) \to D$ satisfying the compatibility constraints is encoded by the membership of these objects to concepts C_d with $d \in D$, while L_v and L_w explicitly represent some of such memberships to encode the initial conditions v and w, respectively. More concretely, L_v and L_w are of the form $\{C_{i,0} \sqsubseteq C_{v_i} \mid i \in \{0, 1, \ldots, n-1\}\}$ and $\{C_{i,0} \sqsubseteq C_{w_i} \mid i \in \{0, 1, \ldots, n-1\}\}$, respectively, where every $C_{i,0}$ is a concept containing exactly the object representing the element $(i, 0) \in U(2^{n+1}, 2^{n+1})$. Let the dl-program $KB = (L, P)$ be defined as follows:

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

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

$$
q \leftarrow \mathit{guess\_ok}^1, \ldots, \mathit{guess\_ok}^l, \mathit{call\_ok}^1, \ldots, \mathit{call\_ok}^l, \tag{29}
$$

and that every P^j_q, $j \in \{1, \ldots, l\}$, is now defined as $P^j_{q,id} \cup P^j_{q,guess} \cup P^j_{q,call}$, where:

1. Every $P^j_{q,id}$, $j \in \{1, \ldots, l\}$, 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:

$$
P^j_{q,id} = \{b^j_i(x) \leftarrow b^{j-1}_i(x) \mid i \in \{0, \ldots, 2l{-}1\},\ i \notin \{2j{-}2, 2j{-}1\}\}.
$$

2. Every $P^j_{q,guess}$, $j \in \{1, \ldots, l\}$, allows for guessing the outcome of the j-th oracle call, that is, exactly one fact among $b^j_{2j-2}(0)$ and $b^j_{2j-2}(1)$. The guess is verified through the predicate $\mathit{guess\_ok}^j$, which

should evaluate to true:

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

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":

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

Hence, 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. □

F Appendix: Proofs for Section 8

Proof of Theorem 8.4. 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 an answer set of KB if and only if A is an answer set of $P \setminus b_U(P) \cup M$, where M is an answer set of $b_U(P)$. The proof is given for the strong answer set semantics; for the weak answer set semantics, it is similar.

($\Rightarrow$) Let A be an 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 an answer set of $b_U(P)$.

Furthermore, A is an 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 can not occur in the body of any rule in $sb_U(P)^A_L$. However, this would contradict that A is an answer set of KB. This shows that A is an answer set of $P \setminus b_U(P) \cup M$, where M is an answer set of $b_U(P)$.

($\Leftarrow$) Let A be an answer set of $R = P \setminus b_U(P) \cup M$ were M is an answer set for $b_U(P)$. Note that $A \supseteq M$, since all literals in M appear in R as facts. The strong reduct 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 can not 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 an 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 an answer set of R. It follows $N = A$. This shows that A is an answer set of KB. □

Proof of Theorem 8.5. 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 $a \rightarrow b$. 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 answer set (if consistent), it is not hard to see that in absence of (possibly) non-monotonic dl-atoms, $(L, b_S(P))$ has some stratification (otherwise), literals $a, b \in S$ must exist such that $a \rightarrow_n b$ and $b \rightarrow^+ a$, which is impossible. In presence of (possibly) non-monotonic 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 and 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, and G. Wagner. Stable model theory for extended RDF ontologies. In *Proceedings ISWC-2005*, volume 3729 of *LNCS*, pages 21–36. Springer, 2005.

[3] J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, and R. Studer. Web Rule Language (WRL), Sept. 2005. W3C Member Submission. http://www.w3.org/Submission/WRL/.

[4] G. Antoniou. Nonmonotonic rule systems on top of ontology layers. In *Proceedings ISWC-2002*, volume 2342 of *LNCS*, pages 394–398. Springer, 2002.

[5] G. Antoniou, C. V. Damásio, B. Grosof, I. Horrocks, M. Kifer, J. Maluszynski, and 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 and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. *J. Autom. Reasoning*, 14(1):149–180, 1995.

[7] C. Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK, 2002.

[8] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[9] T. Berners-Lee. *Weaving the Web*. Harper, San Francisco, CA, 1999.

[10] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[11] H. Boley, S. Tabet, and G. Wagner. Design rationale of RuleML: A markup language for semantic web rules. In *Proceedings SWWS-2001*, pages 381–401, 2001.

[12] P. A. Bonatti, C. Lutz, and F. Wolter. Description logics with circumscription. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 400–410. AAAI Press, 2006.

[13] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 2001.

[14] M. Cadoli and M. Lenzerini. The complexity of propositional closed world reasoning and circumscription. *J. Comput. Syst. Sci.*, 48(2):255–310, 1994.

[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proceedings KR-2006*, pages 260–270. AAAI Press, 2006.

[16] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, chapter 8, pages 229–263. Springer, 2003.

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

[18] J. de Bruijn, T. Eiter, A. Polleres, and H. Tompits. On representational issues about combinations of classical theories with nonmonotonic rules. In *Proceedings KSEM-2006*, volume 4092 of *LNCS/LNAI*, pages 1–22. Springer, 2006. Invited paper.

[19] J. de Bruijn, T. Eiter, A. Polleres, and H. Tompits. Embedding non-ground logic programs into autoepistemic logic for knowledge base combination. In M. Veloso, editor, *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 304–309, Hyderabad, India, January 6–12 2007. AAAI Press/IJCAI.

[20] J. de Bruijn, D. Pearce, A. Polleres, and A. Valverde. A logic for hybrid rules. In *Proceedings Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2006)*. IEEE, 2006. Available at `http://2006.ruleml.org/online-proceedings/rule-integ.pdf`.

[21] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating datalog and description logics. *J. Intell. Inf. Syst.*, 10(3):227–252, 1998.

[22] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. Comput. Log.*, 3(2):177–225, 2002.

[23] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving using the DLV system. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[24] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[25] T. Eiter, G. Gottlob, and H. Veith. Modular logic programming and generalized quantifiers. In *Proceedings LPNMR-1997*, volume 1265 of *LNCS/LNAI*, pages 290–309. Springer, 1997.

[26] T. Eiter, G. Ianni, T. Krennwallner, and R. Schindlauer. Exploiting conjunctive queries in description logic programs. Submitted for publication, Dec. 2006.

[27] T. Eiter, G. Ianni, A. Polleres, R. Schindlauer, and H. Tompits. Reasoning with rules and ontologies. In P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, editors, *Reasoning Web, Second International Summer School 2006, Lissabon, Portugal, September 25-29, 2006, Tutorial Lectures*, number 4126 in LNCS, pages 93–127. Springer, 2006.

[28] T. Eiter, G. Ianni, R. Schindlauer, and 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, and H. Tompits. Nonmonotonic description logic programs: Implementation and experiments. In *Proceedings LPAR-2004*, volume 3452 of *LNCS*, pages 511–517. Springer, 2005.

[30] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proceedings IJCAI-2005*, pages 90–96. Professional Book Center, 2005.

[31] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective integration of declarative rules with external evaluations for Semantic Web reasoning. In *Proceedings ESWC-2006*, volume 4011 of *LNCS*, pages 273–287. Springer, 2006.

[32] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Towards efficient evaluation of HEX programs. In *Proceedings NMR-2006*, pages 40–46, 2006.

[33] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In D. Dubois, C. Welty, and M.-A. Williams, editors, *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), Whistler, British Columbia, Canada*, pages 141–151. AAAI Press, June 2004.

[34] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-founded semantics for description logic programs in the Semantic Web. In *Proceedings RuleML-2004*, volume 3323 of *LNCS*, pages 81–97. Springer, 2004.

[35] W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings JELIA-2004*, volume 3229 of *LNCS/LNAI*, pages 200–212. Springer, 2004.

[36] D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[37] D. Fensel, W. Wahlster, H. Lieberman, and J. Hendler, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002.

[38] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings ICLP/SLP-1988*, pages 1070–1080. MIT Press, 1988.

[39] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[40] M. Gelfond, H. Przymusinska, and T. C. Przymusinski. The extended closed world assumption and its relationship to parallel circumscription. In *Proceedings PODS-1986*, pages 133–139. ACM Press, 1986.

[41] M. Gelfond, H. Przymusinska, and T. C. Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:75–94, 1989.

[42] M. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[43] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In M. Veloso, editor, *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 399–404, Hyderabad, India, January 6–12 2007. AAAI Press/IJCAI.

[44] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, 1992.

[45] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logics. In *Proceedings WWW-2003*, pages 48–57. ACM Press, 2003.

[46] V. Haarslev and R. Möller. RACER system description. In *Proceedings IJCAR-2001*, volume 2083 of *LNCS/LNAI*, pages 701–705. Springer, 2001.

[47] J. Heflin and 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*, pages 63–70. AAAI Press, 1998.

[48] L. A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989.

[49] J. Hendler and D. L. McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.

[50] S. Heymans, D. V. Nieuwenborgh, and D. Vermeir. Nonmonotonic ontological and rule-based reasoning with extended conceptual logic programs. In *Proceedings ESWC-2005*, volume 3532 of *LNCS*, pages 392–407. Springer, 2005.

[51] S. Heymans and D. Vermeir. Integrating ontology languages and answer set programming. In *Proceedings DEXA Workshops 2003*, pages 584–588. IEEE Computer Society, 2003.

[52] S. Heymans and D. Vermeir. Integrating semantic web reasoning and answer set programming. In *Proceedings ASP-2003*, pages 194–208, 2003.

[53] I. Horrocks. DAML+OIL: A description logic for the Semantic Web. *IEEE Bulletin of the Technical Committee on Data Engineering*, 25(1):4–9, 2002.

[54] I. Horrocks. DAML+OIL: A reason-able web ontology language. In *Proceedings EDBT-2002*, volume 2287 of *LNCS*, pages 2–13. Springer, 2002.

[55] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proceedings ISWC-2003*, volume 2870 of *LNCS*, pages 17–29. Springer, 2003.

[56] I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL rules language. In *Proceedings WWW-2004*, pages 723–731. ACM Press, 2004.

[57] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL rules: A proposal and prototype implementation. *J. Web Sem.*, 3(1):23–40, 2005.

[58] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web rule language combining OWL and RuleML, May 2004. W3C Member Submission. http://www.w3.org/Submission/SWRL/.

[59] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. Web Sem.*, 1(1):7–26, 2003.

[60] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings LPAR-1999*, volume 1705 of *LNCS/LNAI*, pages 161–180. Springer, 1999.

[61] J. F. Horty. Some direct theories of nonmonotonic inheritance. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume III: Nonmonotonic Reasoning and Uncertain Reasoning, pages 111–187. Clarendon Press, Oxford, 1994.

[62] U. Hustadt, B. Motik, and U. Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In *Proceedings KR-2004*, pages 152–162. AAAI Press, 2004.

[63] B. Jenner and J. Toran. Computing functions with parallel queries to NP. *Theor. Comput. Sci.*, 141:175–193, 1995.

[64] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2, pages 67–161. MIT Press, Cambridge, MA, 1990.

[65] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings PODS-2002*, pages 233–246. ACM Press, 2002.

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

[67] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[68] V. Lifschitz. Computing circumscription. In *Proceedings IJCAI-1985*, pages 121–127. Morgan Kaufmann, 1985.

[69] V. Lifschitz. Nonmonotonic databases and epistemic queries. In *Proceedings IJCAI-91*, pages 381–386, 1991.

[70] V. Lifschitz and H. Turner. Splitting a logic program. In *Proceedings ICLP-1994*, pages 23–38. MIT Press, 1994.

[71] T. Lukasiewicz. Probabilistic description logic programs. In *Proceedings ECSQARU-2005*, volume 3571 of *LNCS/LNAI*, pages 737–749. Springer, 2005. Extended version in *Int. J. Approx. Reasoning*, in press.

[72] T. Lukasiewicz. Stratified probabilistic description logic programs. In *Proceedings URSW-2005*, pages 87–97, 2005.

[73] T. Lukasiewicz. Fuzzy description logic programs under the answer set semantics for the Semantic Web. In *Proceedings RuleML-2006*, pages 89–96. IEEE Computer Society, 2006.

[74] W. Łukaszewicz. *Non-monotonic Reasoning: Formalizations of Commonsense Reasoning*. Ellis Horwood, 1990.

[75] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and logic programming live together happily ever after? In *Proceedings ISWC-2006*, volume 4273 of *LNCS*, pages 501–514. Springer, 2006.

[76] B. Motik and R. Rosati. A Faithful Integration of Description Logics with Logic Programming. In M. Veloso, editor, *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 477–482, Hyderabad, India, January 6–12 2007. AAAI Press/IJCAI.

[77] B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic ABoxes. In *Proceedings LPAR-2006*, LNCS/LNAI. Springer, 2006.

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

[79] I. Niemelä, P. Simons, and T. Syrjänen. Smodels: A system for answer set programming. In *Proceedings NMR-2000*, 2000.

[80] M. Ortiz de la Fuente, D. Calvanese, T. Eiter, and E. Franconi. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proceedings AAAI-2006*. AAAI Press, 2006.

[81] J. Z. Pan, E. Franconi, S. Tessaris, G. Stamou, V. Tzouvaras, L. Serafini, I. Horrocks, and B. Glimm. Specification of coordination of rule and ontology languages. Project Deliverable D2.5.1, KnowledgeWeb NoE, June 2004.

[82] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[83] A. Polleres, C. Feier, and A. Harth. Rules with contextually scoped negation. In *Proceedings ESWC-2006*, volume 4011 of *LNCS*, pages 332–347. Springer, 2006.

[84] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[85] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

[86] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[87] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[88] R. Rosati. Towards expressive KR systems integrating datalog and description logics: Preliminary report. In *Proceedings DL-1999*, pages 160–164, 1999.

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

[90] R. Rosati. Semantic and computational advantages of the safe integration of ontologies and rules. In *Proceedings PPSWR-2005*, volume 3703 of *LNCS*, pages 50–64. Springer, 2005.

[91] R. Rosati. Integrating ontologies and rules: Semantic and computational issues. In P. Barahona, F. Bry, E. Franconi, N. Henze, and U. Sattler, editors, *Reasoning Web*, volume 4126 of *LNCS*, pages 128–151. Springer, 2006.

[92] R. Rosati. $\mathcal{DL}+log$: Tight integration of description logics and disjunctive datalog. In *Proceedings KR-2006*, pages 68–78. AAAI Press, 2006.

[93] M. Sintek and S. Decker. TRIPLE - A query, inference, and transformation language for the Semantic Web. In *Proceedings ISWC-2002*, volume 2342 of *LNCS*, pages 364–378. Springer, 2002.

[94] T. Swift. Deduction in ontologies via ASP. In *Proceedings LPNMR-2004*, volume 2923 of *LNCS/LNAI*, pages 275–288. Springer, 2004.

[95] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, Germany, 2001.

[96] K. Van Belleghem, M. Denecker, and D. De Schreye. A strong correspondence between description logics and open logic programming. In *Proceedings ICLP-1997*, pages 346–360. MIT Press, 1997.

[97] A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[98] W3C. OWL web ontology language overview, 2004. W3C Recommendation (10 Feb. 2004). Available at http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[99] K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. Merging and aligning ontologies in dl-programs. In *Proceedings RuleML-2005*, volume 3791 of *LNCS*, pages 160–171. Springer, 2005.

[100] K. Wang, D. Billington, J. Blee, and G. Antoniou. Combining description logic and defeasible logic for the Semantic Web. In *Proceedings RuleML-2004*, number 3323 in LNCS, pages 170–181. Springer, 2004.

[101] 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`.

[102] F. Yang, X. Chen, and Z. Wang. p-dl-programs: Combining dl-programs with preference for Semantic Web. Work in progress, Aug. 2006.