Achieving tightness in dl-programs

Luis Cruz-Filipe, Patricia Engrácia, Graça Gaspar and Isabel Nunes

DI–FCUL–TR–2012–03

DOI:10455/6872

(http://hdl.handle.net/10455/6872)

July 2012

Published at Docs.DI (http://docs.di.fc.ul.pt/), the repository of the Department of Informatics of the University of Lisbon, Faculty of Sciences.

Achieving tightness in dl-programs

Luís Cruz-Filipe Patrícia Engrácia Graça Gaspar Isabel Nunes

May 31, 2012

Abstract

In the field of the combination between description logics and rule-based reasoning systems, dl-programs have proved to be a very successful mechanism. One of their recognized shortcomings, however, is their lack of full tightness: the language constructs that feed data from the logic program have a local effect, leaving the knowledge base essentially unchanged throughout. In this paper, we present a construction that we call *lifting*, which allows predicates to be fully shared between the two components of a dl-program in a systematic way, and show how lifting can be used to provide intuitive solutions to a number of everyday reasoning problems involving the verification of integrity constraints and the implementation of default rules. This construction preserves consistency of the underlying knowledge base and complexity of the overall system. Furthermore, the resulting semantics of default rules has a natural interpretation under the original Reiter semantics.

1 Introduction

Description logics (DLs) are extensively used to express ontologies in the Semantic Web due to their nice behaviour – almost all of them are decidable fragments of function-free first-order logic with equality, offering a very good ratio expressiveness/complexity of reasoning [4]. Extending DLs with some kind of rule capability in order to be able to express more powerful queries is a hot topic among the Semantic Web community. Moreover, the adoption of nonmonotonic features (in particular, the negation-as-failure operator not) allows to express default rules and perform closed-word reasoning [16, 15, 19, 14, 28]; the difficult problems it brings, namely in terms of decidability and complexity [27], are worth trying to surpass due to its benefits, and have been one of the main concerns at the base of recent achievements in this area.

In [3] the authors classify existing proposals for integration of rule languages and ontology languages, as *homogeneous* and *heterogeneous* ones. Former approaches combine rules and ontologies in the same logical language, without distinction between rule and ontology predicates, while the latter distinguish them but allow rules to refer to ontology predicates both to access their information and provide them with input.

We chose to work over heterogeneous approaches because we see greater potential for generalization in them: by keeping ontologies and rules separate, they favour independence of ontologies format and semantics, while allowing to retrieve their information and, eventually, feed them with new knowledge. The work we present in this paper goes in the direction of further strengthening these features by e.g. enriching default rules and integrity constraints expression, and allowing for a more fine-grained treatment of the latter.

The structure of the paper is as follows. Section 2 presents the two systems directly related to our presentation: Datalog-based dl-programs and MKNF-knowledge bases, together with criteria for evaluating their success. In Section 3, we discuss an example taken from [25] that captures MKNF's main advantages over dl-programs, presenting some constructions that cannot be directly translated to the latter family of systems. This example will be the underlying motif throughout the whole paper. Section 4 introduces a lifting technique to make dl-programs tight in the sense of [25], and which we use in later sections to support the remainder of our work. Section 5 departs from the example to a more wide-spectrum discussion of integrity constraints in the database world, and shows how we can make dl-programs cope with this kind of specifications. Section 6 deals with default rules, also generalizing from the running example, and discusses their semantics in detail. Both these constructions are shown to preserve decidability and complexity bounds of the systems they are applied to. In Section 7, we

summarize the results obtained, show that they overcome the problems originally presented by the running example, and discuss how dl-programs with lifting compare favourably to systems in the MKNF family. Section 8 discusses other approaches to combining description logics with rule-based formalisms and how other authors have dealt with integrity constraints and default rules. We conclude with some thoughts on the implications of our work and future directions of research.

2 Combining description logics with rules

Combining description logics (DLs) with rules has been a leading topic of research for the past 15 years. In this section, we discuss the approaches that are directly relevant for our work.

First approaches on extending DLs with rules witnessed undesirable results of undecidability when joining decidable fragments of first-order logic (FOL) and logic programming (LP) – in [20] the authors show that, in general, the reasoning problem for a simple description logic combined with an arbitrary Horn logic is undecidable, and identify the constructors that cause the undecidability. For some time, research efforts were put in the direction of finding solutions to this problem. As soon as some were proposed, the main concern became the tractability of reasoning problems.

Decidability Function-free logic programs ensure decidability and termination of evaluation strategies by keeping finite the number of individuals over which the program reasons; in contrast, the decidability of reasoning tasks is guaranteed in most description logics by only considering certain forms of models, e.g. tree-shaped ones whose branching factor is bounded by the size of the knowledge base [4].

The difference in the way these two logics face decidability, leads to undecidabilty problems when they are combined – rules allow us to create undecidable problems by capitalizing over existential quantification in description logics, which can be used to express the existence of infinite chains of objects.

In [20] it is shown that adding arbitrary recursive Datalog rules to even very simple description logics knowledge bases causes undecidability, and two factors – recursion and unsafety of rules – are identified as the ones responsible. The autors propose the concept of *role-safety* – at least one of variables X, Y in a role atom $R(X, Y)$ in a rule r must occur in a predicate in r that does not occur in the head of any program rule.

The goal of combining the DL and LP approaches in the most expressive way while maintaning decidability has been pursued in several ways by the research community, and several other proposals have been presented.

Syntactic restrictions were adopted in cases where the semantics of the interaction of ontologies and rules is tight, that is, where existing semantics for rule languages is adapted directly in the description logic layer. Originally proposed in [14], the *dl-safeness condition* – each rule variable must appear in an atom whose predicate does not occur in the DL knowledge base – controls the interaction between rules and DLs through a syntactic condition on rules. This condition was adopted by several authors, e.g. [26, 27], in several approaches presenting different features – nonrecursive and recursive rules, non-monotonicity, etc –, that eventually adapted it in the sense of reducing loss of expressiveness as, e.g., did [28] with weak dl-safe rules, that is, Datalog rules (see below) where dl-safeness condition is imposed only on the head variables of the rules.

When the integration of ontologies and rules is achieved in a way that keeps a strict separation between the two layers – ontologies are considered as an external knowledge base, with a separated semantics –, syntactic restrictions are not imposed over either part for the purpose of keeping decidability, because reasoning is made in separate, decidable settings, while allowing for a mix of open and closed world reasoning. Examples of this kind of approaches include *dl-programs* [17, 15], which are at the basis of the work presented in this paper.

Complexity The interesting point in what concerns complexity of DL+rules approaches lies in being able to predict what can be expected from the overall complexity of the framework one obtains by applying a specific technique to join a given description logic with a given logic programming language, in terms of the complexities of both languages. In this way, we could make an informed decision when choosing the degree of expressiveness used for reasoning tasks over a given knowledge base.

Complexity results are known for several DL+rules approaches, some of them actually very encouraging, and tractable tools already exist that work well for a great number of cases. Most authors, e.g. [12],

distinguish three main ways to analyse complexity issues in logic programming. Let $\mathcal{P}$ be a function-free logic program, $\mathcal{D}$ be an input database and A be an input ground atom.

- *Data complexity* is the complexity of checking whether $\mathcal{D} \cup \mathcal{P} \models A$ when $\mathcal{P}$ is fixed whereas $\mathcal{D}$ and A are input.
- *Program complexity* (also called expression complexity) is the complexity of checking whether $\mathcal{D} \cup \mathcal{P} \models A$ when $\mathcal{D}$ is fixed whereas $\mathcal{P}$ and A are input.
- *Combined complexity* is the complexity of checking whether $\mathcal{D} \cup \mathcal{P} \models A$ when $\mathcal{D}$, $\mathcal{P}$, and A are all input.

At the end of this section we will discuss complexity results for the two systems at the core of our work.

2.1 dl-programs

Description logic programs or dl-programs [17, 15] are heterogeneous loose coupling combinations of rules and ontologies in the sense that one may distinguish the rule from the knowledge base part, and rules may contain queries to the ontology part of the program. This connection is achieved through *dl-rules*, which are similar to usual rules in logic programs but make it possible, through *dl-atoms* in rule bodies, to query the knowledge base, eventually modifying it (adding facts) for the duration/purpose of the specific query.

For its logic programming part, dl-programs adopted function-free normal logic programs, which can be represented by the well-known Datalog and some of its extensions, used by many successful systems due to its nice properties, namely in what concerns complexity of reasoning tasks. So, before entering in dl-programs details, let us first talk about Datalog.

Datalog is a declarative language based on the logic programming paradigm [11, 31], originally created to be used as a database query language. Datalog programs are function-free collections of rules, which are Horn clauses of the form $\mathsf{Head} \leftarrow \mathsf{Body}$, where Head is an atom, that is, a predicate applied to constants and variables, and Body is a conjunction of zero or more atoms (the subgoals). The variables appearing in the head of a rule are usually called *distinguished variables* in that rule; one says that Head is true for given values of the distinguished variables if there exist values of the non-distinguished variables that make all subgoals of Body true.

In order to guarantee the finiteness of the set of all facts derivable from a Datalog program D, the following safety condition is imposed in every rule: each distinguished and nondistinguished variable appears in at least one nonnegated relational atom in the body of the rule. It follows from this condition that each fact of D is ground.

The semantics that have been proposed for Datalog programs are commonly based on the paradigm of minimal models in which the meaning of a program is given by its least Herbrand model – the set of facts implied by the program –, which contains all the facts in the database, and all the facts that can be derived (directly or indirectly) by the rules, and contains no others.

Several extensions of Datalog exist, from which we emphasize disjunctive Datalog, where the head of the rules may be a disjunction of atoms, and Datalog$^\neg$, which will be used further in this paper, where rules may have negated goals in the body. This negation operator denotes negation-by-failure. Since we do not use disjunctive Datalog, we will not discuss it further.

Reasoning in Datalog$^\neg$ is typically nonmonotonic, that is, it follows the Closed World Assumption (CWA) which determines that all the grounded atoms that cannot be proved from the program facts and rules are assumed to be false. This is consistent with the idea that all relevant information is either: included in the facts database the Datalog program reasons about; or derivable by using the rules.

Going back to dl-programs [17, 15], the introduction of dl-atoms in rules brings the desired interaction between the two worlds, allowing an information flow both from the DL knowledge base to the LP program – given by any query to the DL knowledge base – and vice-versa – given by the input from the LP program that is possible in each query.

A dl-program is a pair $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$, where $\mathcal{L}$ denotes the description logic knowledge base, and $\mathcal{P}$ denotes the generalized logic program – Datalog program extended with dl-atoms in rules. A dl-atom is of the form

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

where each S_i is either a concept, a role, or a special symbol in $\{=, \neq\}$; $op_i \in \{\uplus, \cup\}$, and the symbols p_i are unary or binary predicate symbols depending on the corresponding S_i being a concept or a role. $Q(t)$ is a dl-query, that is, it is either a concept inclusion axiom F or its negation $\neg F$, or of the form $C(t)$, $\neg C(t)$, $R(t_1, t_2)$, $\neg R(t_1, t_2)$, $= (t_1, t_2)$, $\neq (t_1, t_2)$, where C is a concept, R is a role, t, t_1 and t_2 are terms.

In a dl-atom, $p_1, \ldots, p_m$ are called 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. A dl-rule r is a normal rule, i.e. $r = a \leftarrow b_1, \ldots, b_k, \mathbf{not}\, b_{k+1}, \ldots, \mathbf{not}\, b_m$, where any $b_1, \ldots, b_m$ may be a dl-atom. We denote the head a of r by $H(r)$, its positive atoms $b_1, \ldots, b_k$ by $B^+(r)$, and its negated atoms $b_{k+1}, \ldots, b_m$ by $B^-(r)$.

The authors of [17, 15] integrated the underlying logics of OWL LITE and OWL DL with normal logic programs, extending two different semantics for ordinary normal programs – answer-set semantics and well-founded semantics. In this paper, we will work with the latter, which those authors define by generalizing the well-founded semantics for ordinary normal programs. We first recall the notion of unfounded sets for dl-programs.

Given a dl-program $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$, we denote by $Lit_\mathcal{P}$ the set of all literals from P. Let $I \subseteq Lit_\mathcal{P}$ be consistent. A set $U \subseteq HB_\mathcal{P}$, where $HB_\mathcal{P}$ is $\mathcal{P}$'s Herbrand base, is an *unfounded set* of $\mathcal{KB}$ relative to I iff for every $a \in U$ and every ground rule r with $H(r) = a$, one of the following holds.

(i) $\neg b \in I \cup \neg U$ for some ordinary atom $b \in B^+(r)$

(ii) $b \in I$ for some ordinary atom $b \in B^-(r)$

(iii) For some dl-atom $b \in B^+(r)$, it holds that $S^+ \not\models_\mathcal{L} b$ for every consistent $S \subseteq Lit_\mathcal{P}$ with $I \cup \neg U \subseteq S$

(iv) For some dl-atom $b \in B^-(r)$, $I^+ \models_\mathcal{L} b$

Unfounded sets are important to "build" the well-founded semantics of a dl-program $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$, since they are used in the definition of the operators that support it. The $T_\mathcal{KB}$, $U_\mathcal{KB}$ and $W_\mathcal{KB}$ operators, on all consistent $I \subseteq Lit_\mathcal{P}$, are as follows:

- $a \in T_\mathcal{KB}(I)$ iff $a \in HB_\mathcal{P}$ and some ground rule r exists such that all of the following hold.
  - (i) $H(r) = a$
  - (ii) $I^+ \models_\mathcal{L} b$ for all $b \in B^+(r)$
  - (iii) $\neg b \in I$ for all ordinary atoms $b \in B^-(r)$
  - (iv) $S^+ \not\models_\mathcal{L} b$ for each consistent $S \subseteq Lit_\mathcal{P}$ with $I \subseteq S$, for all dl-atoms $b \in B^-(r)$
- $U_\mathcal{KB}(I)$ is the greatest unfounded set of $\mathcal{KB}$ relative to I (it exists because the set of unfounded sets of $\mathcal{KB}$ relative to I is closed under union); and
- $W_\mathcal{KB}(I) = T_\mathcal{KB}(I) \cup \neg U_\mathcal{KB}$.

These operators are all monotonic thus, in particular, $W_\mathcal{KB}$ has a least fixpoint, denoted $lfp(W_\mathcal{KB})$. The well-founded semantics for dl-programs – $WFS(\mathcal{KB})$ – is defined as being $lfp(W_\mathcal{KB})$.

From the complexity point of view, encouraging results were proved for dl-programs under well-founded semantics, as will be discussed ahead.

In [19] the concept of Datalog-rewritable description logic ontologies is introduced defining the class of ontologies that can be rewriten to Datalog programs, such that dl-programs can be reduced to Datalog with negation, under well-founded semantics, allowing for tractable DL reasoning via Datalog engines.

Unfortunately, the syntactic restrictions Datalog-rewritability imposes are not met by the expressiveness requirements our proposal implies. However, as will be presented, worst-case complexity of dl-programs is preserved.

2.2 MKNF and related systems

The logic of minimal knowledge and negation as failure (MKNF) was proposed in [21] as a new version, not restricted to the propositional case, of the logic of Grounded Knowledge (GK) [22], which gives a simple, epistemic semantics for both Reiter's Default Logic and Moore's Autoepistemic Logic. A close

relation between each of these two logics and negation as failure was proposed in [7] – where negated atoms in the bodies of rules are treated as justifications (the rule $s \leftarrow p, \mathbf{not}\, q$, for instance, can be understood as the default that allows us to derive s from p assuming that $\neg q$ is consistent) –, and in [18] – where $\mathbf{not}\, p$ in the body of a rule can be read as "p is not believed", thereby understanding a rule with negation as the corresponding formula of autoepistemic logic.

MKNF is an extension of first-order logic with modal operators $\mathbf{K}$ and $\mathbf{not}$, whose formulae syntax is defined by the following grammar, where t_i are first-order terms and P is a predicate:

$$\varphi ::= \mathbf{true} \mid P(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists x : \varphi \mid \mathbf{K}\,\varphi \mid \mathbf{not}\,\varphi$$

The operators $\vee$ and $\subset$ are defined as syntactic shortcuts. Intuitively, $\mathbf{K}$ is an S5-modality and $\mathbf{not}$ corresponds to $\neg\,\mathbf{K}$. The formal semantics for MKNF, however, is more complicated than the semantics for usual first-order modal logic, and we will not go into details here since it will not be relevant for our discussion.

An MKNF formula of the following form is called a rule, as it generalizes the notion of a rule in disjunctive logic programs interpreted under stable model semantics:

$$\mathbf{K}\,H_1 \vee \ldots \vee \mathbf{K}\,H_k \subset \mathbf{K}\,B_1 \ldots \wedge \mathbf{K}\,B_m \wedge \mathbf{not}\,B_{m+1} \wedge \ldots \wedge \mathbf{not}\,B_{m+n}$$

where Hs and Bs are head and body literals, i.e., of the form A and $\neg A$ where A is a first-order atom.

In [25], the authors extend MKNF, which has been designed to allow embeding of DL knowledge bases as well as first-order rules, with the necessary features that make it possible to use it as a very general semantic framework for an integration of description logics (as vehicles to specify and reason about open-world conceptual knowledge) with rules (as a means to provide for nonmonotonic reasoning). The hybrid formalism of MKNF$^+$ knowledge bases is proposed in order to enable the required mix of open and closed world reasoning that is necessary to express, e.g., integrity constraints and default rules, and shown to be general enough to capture several existing languages and approaches.

Let $\mathcal{B}$ be a generalized atom base, i.e. a set of first-order formulae such that $\varphi \in \mathcal{B}$ implies $\varphi_G \in \mathcal{B}$ for each grounding φ_G of φ. An MKNF$^+$ rule over $\mathcal{B}$ is a formula of the form

$$H_1 \vee \ldots \vee H_k \leftarrow B_1, \ldots, B_n$$

where each H_i is a nonmodal or a $\mathbf{K}$-atom over $\mathcal{B}$ and each B_i is either a nonmodal or a $\mathbf{K}$-atom or a $\mathbf{not}$-atom over $\mathcal{B}$. A rule is safe if each variable that occurs free in some rule atom also occurs free in a body $\mathbf{K}$-atom. An MKNF$^+$ knowledge base over $\mathcal{DL}$ and $\mathcal{B}$, where $\mathcal{DL}$ is a description logic and $\mathcal{B}$ is a generalized atom base, is a pair $\mathcal{K} = \langle \mathcal{O}, \mathcal{P} \rangle$ where $\mathcal{O} \in \mathcal{DL}$ is a knowledge base and $\mathcal{P}$ is a program over $\mathcal{B}$.

In [25] the authors explain that the existence of both modal and nonmodal atoms adds complexity to this system, because it mixes the first-order with the nonmonotonic aspects of reasoning, and they introduce a restricted version of MKNF$^+$ knowledge bases – MKNF knowledge bases – in which each atom in each rule must be a $\mathbf{K}$-atom or a $\mathbf{not}$-atom. They further define the conditions that guarantee the existence of a given MKNF$^+$ knowledge base's restricted version. In MKNF knowledge bases it is possible to identify first-order reasoning with formulae both in $\mathcal{DL}$ and in an appropriate extension of the generalized atom base $\mathcal{B}$, and to identify nonmonotonic reasoning with the modal MKNF rules.

2.3 Comparing the two approaches

The main difference between MKNF-related systems and dl-programs is whether the description logic and the logic program are kept separate. Indeed, dl-programs are examples of a *heterogeneous* combination of systems: both components are kept completely apart, albeit communicating, and their reasoning mechanisms are maintained. This has a number of advantages. First, both components can be studied and developed independently – useful in real-life systems, where the description logic component can be an ever-evolving ontology. Secondly, the clean separation of two different worlds automatically avoids several technical problems related to decidability and complexity, bypassing syntactic constraints such as dl-safety. This is very obvious in example formalizations presented for both systems: *homogeneous* MKNF formalizations such as the one we replicate in Section 3 must resort to *ad hoc* conventions such as the use of upper/lowercase predicate names to enhance readability, while dl-queries in dl-programs allow for no ambiguity about the origin of predicates and the kind of reasoning being used.

Heterogenous approaches also tend to come with preservation results, such as decidability and complexity bounds: since dl-queries are atomic formulas, the complexity bounds for the logic program component are unaffected, while the processing of a dl-query is essentially independent of the context it originated from. Of course this is a simplification, since the feedback mechanisms described above may require some extra computation; but one quickly gets a feeling for the global understanding of the combined system. In homogeneous approaches one always ends up having to reduce two different systems to the same syntactical setting, which typically implies reencoding and reanalysing at least one of them.

Indeed, [15] presents a number of encouraging results for dl-programs: (i) data complexity is preserved for dl-programs reasoning under well-founded semantics (complete for P) as long as dl-queries in the program can be evaluated in polynomial time, and (ii) reasoning for dl-programs is first-order rewritable (and thus in LOGSPACE under data complexity) when the evaluation of dl-queries is first-order rewritable and the dl-program is acyclic. This nice results are consequence of the use of Datalog: complexity of query evaluation on both Datalog and stratified Datalog$^\neg$ programs is data complete for PTIME and program complete for EXPTIME, as shown in [12].

On the other hand, complexity results for MKNF knowledge bases under the dl-safety restriction, presented in [25], state that data complexity of entailment checking is coNP even when there is no underlying description logic as long as default negation is available. This complexity bound holds also when the DL component is PTIME-complete.

However, [25] gives a list of other criteria by means of which different approaches to combining description logics and rule-based approaches should be compared, thereby justifying the development of MKNF knowledge bases as a system that has all the desirable properties while being general enough to embrace many other existing systems. These are the following four aspects: *faithfulness* (the semantics of both components is preserved), *tightness* (both components contribute to the consequences of one another), *flexibility* (the same predicate can be viewed under closed- and open-world interpretation at will) and *decidability*. MKNF knowledge bases are faithful, tight, flexible and – as long as the syntactic requirements described above are respected – decidable; dl-programs, the authors argue, are faithful, flexible and decidable, but not tight. The latter is not completely true: the possibility of dynamically extending extents of predicates in a dl-query means that the logic program effectively sees an extended knowledge base for the purpose of that query, hence some (restricted, local) form of tightness is available. This is actually much more versatile, since one can choose for *each* query how each predicate should be extended. Universally extending a predicate (thereby achieving true tightness) can be realized by means of the constructions we introduce in Section 4.

The authors of [25] also show how MKNF can formalize several useful constructions that are not easy to implement in heterogeneous systems. The goal of our paper is to show how these constructions can be implemented in dl-programs, thus retaining these systems' good properties, and at the same time bringing them closer to MKNF in terms of expressivity.

3 The problem

We will focus on an example by Motik et al. (Section 3.5 of [25]) to motivate the constructions we propose. The example deals with a system designed to help users decide where to go on holidays, including some basic facts about several European cities and some reasoning rules to derive possible holiday destinations.

Since most of the formalization in MKNF is directly translatable to a dl-program, we present side-by-side in Table 3 both Motik's formalization and the corresponding dl-program versions. We follow the standard naming conventions for each system. Thus, in MKNF (left column) predicate names from the description logic start with lowercase, whereas predicate names from the logic program start with uppercase; constants start with uppercase, while variables are lowercase single letters. In dl-programs (right column) the convention is the standard from logic programming: predicate names and constants all start with lowercase (and from the context one can determine whether the predicate comes from the description logic component or from the logic program), while variables are uppercase letters.

The problematic rules in the translation are starred.

Rules (1–6) are simply facts included in the description logic's A-Box, hence they are translated directly. Similarly, rules (7–8) belong in the description logic's T-Box and are also directly translated. Note that, in order to allow rule (7), this description logic must be more expressive than Eiter's $\mathcal{LDL}^+$, and hence might not be Datalog-rewritable [19].

	MKNF	dl-program
(1)	portCity(Barcelona)	portCity(barcelona)
(2)	onSea(Barcelona, Mediterranean)	onSea(barcelona, mediterranean)
(3)	portCity(Hamburg)	portCity(hamburg)
(4)	¬seasideCity(Hamburg)	¬seasideCity(hamburg)
(5)	rainyCity(Manchester)	rainyCity(manchester)
(6)	has(Manchester, AquaticsCenter)	has(manchester, aquaticsCenter)
(7)	seasideCity $\sqsubseteq \exists$has.beach	seasideCity $\sqsubseteq \exists$has.beach
(8)	beach $\sqsubseteq$ recreational	beach $\sqsubseteq$ recreational
(9)	**K** seasideCity(x) $\leftarrow$ **K** portCity(x),**not** ¬seasideCity(x)	$(*)$
(10)	**K** InterestingCity(x) $\leftarrow$ **K** [$\exists y$:has(x,y) $\wedge$ recreational(y)], **not** rainyCity(x)	interestingCity(X) $\leftarrow$ DL[;$\exists$has.recreational](X), **not** DL[;rainyCity(X)]
(11)	**K** HasOnSea(x) $\leftarrow$ **K** onSea(x,y)	$(**)$
(12)	false $\leftarrow$ **K** seasideCity(x), **not** HasOnSea(x)	
(13)	**K** SummerDestination(x,y) $\leftarrow$ **K** InterestingCity(x), **K** onSea(x,y)	summerDestination(X,Y) $\leftarrow$ interestingCity(X),DL[;onSea](X,Y)

Table 1: An MKNF example and the corresponding dl-program.

Rule (9) presents the first serious problem. This rule extends the set of terms t for which seasideCity(t) holds provided some consistency requirements are met. However, when we try to write this as a dl-program rule we stumble upon a technical impossibility: seasideCity is a concept (predicate) from the description logic, but this rule should be implemented as part of the logic program. To get around this issue, we first observe that this is an instance of a default rule; in Section 6 we present a way to deal with default rules in general, which overcomes this particular apparent impossibility. Note that [16, 13] already introduced a way to model default rules in dl-programs; our approach has some similarities, but is a particular application of a more general construction, and therefore of interest in itself. Also, we will discuss theoretical aspects of this representation that have not, to our knowledge, been approached before.

Rule (10) is straightforward, as long as one identifies where each predicate in the premise comes from. Predicates from the description logic must be enclosed in a dl-query. Comparing both versions of the rule highlights the clear distinction in dl-programs between the two universes (the description logic and the logic program) that is absent from the MKNF formalization.

Rules (11–12) present another problem. These two rules act together as an integrity constraint, requiring that, for every city known to be at the seaside, information be present in the description logic about which sea bathes the city. Although rule (11) in itself is not problematic, rule (12) is difficult to translate since typical logic programs do not allow false as head of a rule. In Section 5 we discuss how this type of integrity constraint can be expressed in dl-programs in a not-so-direct (but, we argue, more satisfactory) way.

Finally, rule (13) is again straightforward to translate. Note the (questionable) inclusion of y in the head of the MKNF knowledge base rule, needed in order to ensure that the rule is dl-safe but not very intuitive from a semantic point of view. Although the dl-program version we present for this rule also includes this variable, this is not necessary because DL-safety of rules is not a condition to achieve decidability in dl-programs.

The structure of the remaining sections is as follows. Section 4 introduces a seemingly inocuous way to lift predicates from the description logic to the level of the logic program so that they can be dynamically extended in a systematic way. This construction is used in Sections 6 and 5 to develop systematic ways to deal with specific kinds of default rules and integrity constraints that, in particular, allow us to formalize this example completely. We also show that (1) these constructions preserve the consistency of the dl-program and (2) these constructions preserve the worst-case complexity of reasoning in these dl-programs. Of course, the last result is not as impressive as it may seem, since the expressiveness requirements of the description logic underlying the program prevent the use of e.g. datalog-rewritable description logics. We also analyze in some detail the precise semantics of our encoding of default rules.

4 Sharing predicates

The first main difference between MKNF and dl-programs is the possibility (in MKNF) of sharing predicates between the database component and the logic program component of the logic. In MKNF, this is not an issue, since these components are not kept separate; however, in dl-programs there is a strict distinction between the database part (the underlying description logic knowledge base $\mathcal{L}$) and the reasoning part (the generalized logic program $\mathcal{P}$).

To deal with this, we need a way to "upgrade" $\mathcal{L}$-predicates to $\mathcal{P}$. We do this as follows.

Definition 1. Let $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ be a dl-program, where $\mathcal{P}$ is a generalized Datalog$^\neg$ program[1] and let Q be an $\mathcal{L}$-predicate.[2] The *lifting* of Q to $\mathcal{P}$ is achieved by:

(i) enriching $\mathcal{P}$ with two fresh predicate names q^+ and q^-, which will intuitively correspond to Q and $\neg Q$, together with the rules

$$q^+(X) \leftarrow DL[Q \uplus q^+, Q \uplus q^-; Q](X) \qquad q^-(X) \leftarrow DL[Q \uplus q^+, Q \uplus q^-; \neg Q](X) \tag{1}$$

(ii) replacing every dl-atom $DL[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n; R](t)$ with

$$DL[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n, Q \uplus q^+, Q \uplus q^-; R](t)\,. \tag{2}$$

We will denote the latter query as $DL_{\{Q\}}[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n; R](t)$, and refer to it as the *extended* version of the original query; if several (different) predicates are being lifted – a situation we will discuss later on – we will write $DL_{\{Q_1,\ldots,Q_k\}}[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n; R](t)$ with the expected meaning. Note that the order in which different predicates are lifted is irrelevant, since description logics deal with sets of formulas, so this notation is unambiguous.

Observe that the description logic must be sufficiently expressive for this construction to work; in particular, it must have (classical) negation. This is not a serious drawback, since most of the logics underlying dl-programs do have negation, but it has some impact on the overall complexity of the logic.

Lemma 1. Let $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ be a dl-program and $\mathcal{KB}_{\{Q\}}$ be the same dl-program enriched with the lifting of some $\mathcal{L}$-predicate Q. For every dl-atom $DL[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n; R](t)$ in $\mathcal{KB}$, if the knowledge base underlying its query is consistent, then so is the knowledge base underlying the query $DL_{\{Q\}}[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n; R](t)$ in $\mathcal{KB}_{\{Q\}}$.

By "knowledge base underlying the query" $DL[S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n; R](t)$ we mean the description logic knowledge base $\mathcal{L}$ where the predicates $S_1, \ldots, S_n$ have been adequately extended according to the definition of $op_1, \ldots, op_n$.

Proof. The result follows directly from the definition: since q^+ and q^- are fresh in $\mathcal{P}$, the only way to prove e.g. $q^+(t)$ is by applying the rule

$$q^+(X) \leftarrow DL[Q \uplus q^+, Q \uplus q^-; Q](X)\,,$$

whose premise can only be satisfied if $Q(t)$ already holds. Hence the extents of Q and $Q \uplus q^+$ actually coincide. The case for $Q \uplus q^-$ is similar. □

By *extent* of a predicate Q we mean the set of terms t such that $Q(t)$ holds.

At this point, the reader might wonder what the purpose of extending Q (and $\neg Q$) with q^+ (and q^-) is, since these predicates add nothing new to the knowledge base. The point is that lifting identifies a predicate in $\mathcal{L}$ with a predicate in $\mathcal{P}$, allowing one to reason about it in both contexts. In this way, we can e.g. add closed-world reasoning about a predicate originally defined in $\mathcal{L}$ (where the semantics is open-world). In the next sections, we will introduce some specific constructions in dl-programs, such as adding default rules and dealing with integrity constraints, that become very elegant to express capitalizing on the notion of lifting.

[1] Throughout this paper, we will assume that $\mathcal{P}$ is always a generalized Datalog$^\neg$ program and refrain from stating this explicitly.

[2] We will use an uppercase letter for single-letter generic predicate names from the description logic to make reading easier.

Lifting is a step that must be done prior to the addition of any new reasoning rules, since there is *a priori* no way to identify $\mathcal{L}$ predicates with predicates from $\mathcal{P}$; the theorem stated above is relevant because it states that lifting is effectively inoffensive, since it adds no new information to the program. There is however a small catch: rule (2) must hold even for dl-queries that are added to the program later on. Therefore, we define a more general, time-independent, notion.

Definition 2. The *dl-program with lifting* $\mathcal{KB}_\Gamma$, where $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ is a dl-program and $\Gamma = \{Q_1, \ldots, Q_m\}$ is a finite set of $\mathcal{L}$-predicates, is the dl-program $\langle \mathcal{L}, \mathcal{P}_\Gamma \rangle$ where $\mathcal{P}_\Gamma$ is obtained from $\mathcal{P}$ by:

- for every $Q \in \Gamma$, adding the rules

$$q^+(X) \leftarrow DL[;Q](X) \qquad\qquad q^-(X) \leftarrow DL[;\neg Q](X) \tag{3}$$

- replacing every dl-query $DL[\chi; R](t)$ (including those added in the previous step) with

$$DL[\chi, Q_1 \uplus q_1^+, Q_1 \uplus q_1^-, \ldots, Q_m \uplus q_m^+, Q_m \uplus q_m^-; R](t) \tag{4}$$

  where $\chi \equiv S_1\ op_1\ p_1, \ldots, S_n\ op_n\ p_n$ corresponds to the original query's input. We will call the query in (4) a Γ-*extended query* and abbreviate it to $DL_\Gamma[\chi; R](t)$.

Notice that the notation $DL_\Gamma[\chi; R](t)$ is consistent with the one introduced after the previous definition. In practice, one can define a dl-program with lifting simply by giving $\mathcal{KB}$ and Γ. We will work in this way when convenient; however, the semantics of a dl-program with lifting are defined over $\mathcal{P}_\Gamma$, so whenever we discuss these semantics we will write $\mathcal{P}_\Gamma$ explicitly.

Restating Lemma 1 in terms of this definition, we obtain the following result.

Theorem 1. Let $\mathcal{KB}_\Gamma = \langle \mathcal{L}, \mathcal{P}_\Gamma \rangle$ be a dl-program with lifting where no rule in $\mathcal{P}$ uses a lifted predicate name in its head. For every dl-atom in $\mathcal{P}$, if the knowledge base underlying its query in $\mathcal{KB}$ is consistent, then so is the knowledge base underlying it in $\mathcal{KB}_\Gamma$.

The restriction in Theorem 1 is essential, since we no longer use fresh predicate names (because we do not want to distinguish "before lifting" and "after lifting", but rather to see lifting as part of the program), and rules using lifted predicates can easily make the knowledge base underlying extended queries inconsistent – just consider the case where $\mathcal{P}$ contains the rule $q^+ \leftarrow q^-$. This establishes the following good programming practice for dl-programs with lifting: one should take care to add Q to Γ before enriching $\mathcal{P}$ with any rules involving q^+ or q^-.

From the proof of Lemma 1 one also obtains the following result, stating that indeed Q and q^+ have the same semantics.

Corollary 1. In the conditions of Theorem 1, the sets $\{t \mid Q_i(t)\}$ and $\{t \mid q_i^+(t)\}$ coincide for every i, as well as the sets $\{t \mid \neg Q_i(t)\}$ and $\{t \mid q_i^-(t)\}$.

It is simple to show that these relationships remain valid after adding new rules to the logic program as long as $\{t \mid Q_i(t)\}$ and $\{t \mid \neg Q_i(t)\}$ are interpreted as in an extended query, so Corollary 1 holds in general. In other words, q_i^+ and q_i^- are true counterparts to Q_i and $\neg Q_i$, tightly brought together by the lifting construction.

Besides providing a way to make dl-programs tight, as we will discuss in Section 7, lifting is very useful (and even, in some cases, essential) to model some specific constructions. The next sections discuss some examples in detail.

5 Integrity constraints in dl-programs

One of the limitations of dl-programs pointed out in [25] is that formalism's inability to express database integrity constraints. Database dependencies have been since long a main tool in the fields of relational and deductive databases [1], used to express integrity constraints on databases. They formalize relationships between data in the database that need to be satisfied so that the database conforms to its intended meaning.

In an effort to generalize different forms of dependencies, Beeri and Vardi [6] defined two types of dependencies that comprehend most of the database dependencies found in the literature: tuple-generating dependencies and equality-generating dependencies. A *tuple-generating dependency* is a first-order formula

$$\forall \overline{x}.(\varphi(\overline{x}) \rightarrow \exists \overline{y}.\psi(\overline{x},\overline{y}))$$

where φ and ψ are conjunctions of relation atoms and φ uses all the variables in $\overline{x}$. An *equality-generating dependency* is a first-order formula

$$\forall \overline{x}.(\varphi(\overline{x}) \rightarrow (x_i = x_j))$$

where x_i and x_j are variables from $\overline{x}$.

A third type of database dependencies commonly found in the literature consists of *negative constraints*

$$\forall \overline{x}.\,(\varphi\,(\overline{x}) \rightarrow \bot)$$

where $\bot$ corresponds to the truth constant false.

None of these types of dependencies can be expressed as dl-rules, since they would require the use of, respectively, an existential quantification, an equality atom, and $\bot$ in their head. Therefore, several approaches have been proposed recently to extend either OWL or dl-programs to be able to express integrity constraints, as is discussed in more detail in Section 8. In particular, the way to formalize tuple-generating dependencies in MKNF proposed in [25], which we already presented in Section 3, is to include some rules that will lead to inconsistency whenever these constraints are not met.

Let us consider a simple example, taken from p. 9 of [25].

Example 1. Consider a setting where we want to model some common knowledge about people; in particular, we want to state that every person has a social security number (SSN)

$$\forall x.(\mathsf{person}(x) \rightarrow \exists y.\mathsf{hasSSN}(x,y))\,.$$

In the cited paper, the authors formalize this in an MKNF knowledge base in two steps. Since the head of the rule may not have existencial quantifiers, first they add a new predicate $\mathsf{SSN\_OK}(x)$; then they write down the constraint as the following two rules.

$$\begin{aligned} \mathsf{SSN\_OK}(x) &\leftarrow \mathsf{hasSSN}(x,y) \\ \mathsf{false} &\leftarrow \mathsf{person}(x), \mathsf{not}\,\mathsf{SSN\_OK}(x) \end{aligned}$$

This approach is not possible in typical dl-programs, since the heads of rules in (generalized) logic programs may not include $\bot$. However, we feel that this is not even desirable: the fact that the database is inconsistent should be flagged, but by no means does it imply that falsity is derivable. Hence, we propose an alternative way to do this, using special (fresh) inconsistency predicates that can be used to check whether specific integrity constraints hold at some point in time. Note that this technique corresponds to standard practice in the logic programming community.

Let $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ be a dl-program, where $\mathcal{L}$ models the common knowledge we want to reason about. In order to write down the same integrity constraint, we can add the following dl-rules, where $\bot_{\mathsf{personSSN}}$ is simply a (fresh) predicate.

$$\begin{aligned} \bot_{\mathsf{personSSN}}(X) &\leftarrow DL[;\mathsf{person}](X), \mathsf{not}\,\mathsf{ssn\_ok}(X) \\ \mathsf{ssn\_ok}(X) &\leftarrow DL[;\mathsf{hasSSN}](X,Y) \end{aligned}$$

The integrity check can now be performed by specifying the goal $\leftarrow \bot_{\mathsf{personSSN}}(X)$. If this succeeds, then there must be some person in $\mathcal{L}$ for which hasSSN does not hold, and the database is inconsistent in the sense that it does not satisfy the required integrity constraint, unlike in the solution proposed in [25], where $\leftarrow \mathsf{false}$ will succeed if *any* integrity constraint fails to hold, and it is not possible to know *which* it was without further examination. Furthermore, the user can uniquely identify where the inconsistency resides (the instantiation found for X), instead of just receiving an answer that the integrity constraint does not hold. This is due to the use of a unary inconsistency check predicate.

Notice that in order just to check this integrity constraint, no lifting is necessary since predicates person and hasSSN are not extended. However, if these rules are part of a larger program $\mathcal{P}$, then they may become part of a dl-program obtained from lifting some $\mathcal{L}$-predicates to $\mathcal{P}$.

Negative constraints can be checked in a similar way.

Example 2. Let $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ be a dl-program where $\mathcal{L}$ models some common knowledge about vegetarian people. We want to add a constraint stating that no vegetarian eats meat, corresponding to the following negative constraint, where vegetarian and meat are concepts and eats is a role in $\mathcal{L}$.

$$\forall xy.((\mathsf{vegetarian}(x) \wedge \mathsf{eats}(x, y) \wedge \mathsf{meat}(y)) \rightarrow \bot)$$

We can add the following dl-rule, where $\bot_{\mathsf{vegetarian}}$ is the new inconsistency predicate

$$\bot_{\mathsf{vegetarian}}(X, Y) \leftarrow DL[; \mathsf{vegetarian}](X),\ DL[; \mathsf{eats}](X, Y),\ DL[; \mathsf{meat}](Y)$$

The integrity check can now be performed by specifying the goal $\leftarrow \bot_{\mathsf{vegetarian}}(X, Y)$. Again, if this goal succeeds we will not only know that this integrity constraint is not being satisfied, but we will also get an answer detailing the reason why this is the case.

Notice that the integrity check predicate now has two free variables. In general, this predicate will have as many variables as the universal quantifier in the corresponding logic formula: they are the instances that characterize the facts in the knowledge base violating the constraint.

The next example illustrates the use of dl-rules to check a full tuple-generating dependency, that is, a tuple-generating dependency that has no existencial quantifier.

Example 3. Let $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ be a dl-program where $\mathcal{L}$ models some knowledge about courses, their lecturers and recommended books. We want to add a constraint stating that whenever a lecturer teaches a course for which a certain book is recommended, that lecturer must use that book. This corresponds to the following full tuple-generating dependency.

$$\forall xyz.(\mathsf{teachesCourse}(x, y) \wedge \mathsf{recommendsBook}(y, z) \rightarrow \mathsf{usesBook}(x, z))$$

There are three universally quantified variables in this rule, so we now use a ternary integrity check predicate $\leftarrow \bot_{\mathsf{usesBook}}$, and add the following dl-rule to the program.

$$\bot_{\mathsf{usesBook}}(X, Y, Z) \leftarrow DL[; \mathsf{teachesCourse}](X, Y), DL[; \mathsf{recommendsBook}](Y, Z), \mathsf{not}\ DL[; \mathsf{usesBook}](X, Z)$$

In this particular situation, however, there is another, potentially more interesting way to deal with this problem. The role of an integrity constraint is usually as a check to the knowledge base, and that was the view we adopted so far. But in this case the integrity constraint can be used to infer new facts that "update" the knowledge base and make it satisfy that constraint.

There is a small catch: we cannot place a dl-atom as the head of the rule. The answer, of course, is to apply lifting to the relevant predicate (so we work with a dl-program with lifting where $\Gamma = \{\mathsf{usesBook}\}$) and write the rule as follows.

$$\mathsf{usesBook}^{+}(X, Z) \leftarrow DL[; \mathsf{teachesCourse}](X, Y), DL[; \mathsf{recommendsBook}](Y, Z) \tag{5}$$

Recall from the definition of dl-program with lifting that the underlying dl-program P_Γ will also have the two rules

$$\mathsf{usesBook}^{+}(X, Y) \leftarrow DL_\Gamma[; \mathsf{usesBook}](X, Y)$$
$$\mathsf{usesBook}^{-}(X, Y) \leftarrow DL_\Gamma[; \neg\mathsf{usesBook}](X, Y)$$

and furthermore all queries, namely the two queries in rule (5), are replaced with the extended queries according to (4).

When dealing with full tuple-generating dependencies, both options are acceptable as ways of formalizing integrity constraints. There are advantages and drawbacks to either one, so the decision between them must be made on a case-by-case basis. On the one hand, using the information provided by the integrity constraint to complete the database via lifting is an interesting possibility, since it allows one automatically to correct some problems, instead of just providing the information that they are there. However, from the moment we start adding rules that may extend lifted predicates there is no guarantee that the knowledge base remains consistent; so the first implementation of the integrity constraint is the only one that is always safe to use.

This approach allows us to maintain unchanged the original knowledge base $\mathcal{L}$ that uses open-world semantics and typically has incomplete data, yet enrich it differently in applications that may benefit from different treatments of integrity constraints. Depending on the rules in $\mathcal{P}$ and on the set Γ of lifted predicates, we may simply check the satisfaction of an integrity constraint or we may enforce the update of a certain predicate in order to ensure that a constraint is satisfied.

The repair of databases so that they are minimally changed to ensure the satisfaction of integrity constraints is the subject of much work in the topics of active integrity constraints and revision programming [10]. For description logic knowledge bases, lifting may provide a first step in the direction of further extensions of dl-programs to deal fully with repairing issues. That is, however, outside the scope of this report.

Finally, we mention that in the context of dl-programs equality-generating dependencies are a particular case of full tuple-generating dependencies, since equality (and non-equality) are primitive predicate symbols in the majority of description logics. However, lifting equality does not look like a good idea, since it is also typically a special predicate with special semantics. Therefore, equality-generating dependencies should be always treated only as checks (in the sense of the first examples), implemented as full tuple-generating dependencies whose dl-queries happen to use equality.

6 Encoding default rules in dl-programs

A default rule (à la Reiter) is a triple $\langle \{\varphi_1, \ldots, \varphi_n\}, \{\psi_1, \ldots, \psi_m\}, \theta \rangle$ with the following intended semantics: if there is some instantiation of the free variables in the rule such that (1) $\varphi_1, \ldots, \varphi_n$ all hold and (2) it is consistent to assume that $\psi_1, \ldots, \psi_m$ also hold simultaneously, then we may infer θ. We will use an inference rule-like notation for default rules, i.e. the previous rule will be written as

$$\frac{\varphi_1, \ldots, \varphi_n : \psi_1, \ldots, \psi_m}{\theta}$$

In [2] the semantics of Default Logic is given in terms of *extensions* which intuitively represent expansions of the set of known facts with reasonable assumptions w.r.t. the given defaults. The author presents an operational definition of extensions – process tree semantics –, which we reproduce here since it is the semantics for default logic we will use. Although this semantics operates in a different way from Reiter's original semantics, they both rely on equivalent notions of extension; furthermore, the semantics in [2] also captures other important concepts that we will introduce when necessary.

6.1 Extension semantics for Default Logic

Given a default theory $\mathcal{T} = (\mathcal{W}, \mathcal{D})$, where $\mathcal{W}$ is a set of facts or axioms and $\mathcal{D}$ is a countable set of default rules, or defaults, the following notions are needed for the operational definition of extensions:

- given a default rule δ of the form $\dfrac{\varphi : \psi_1, \ldots, \psi_m}{\theta}$, $pre(\delta) = \varphi$, $just(\delta) = \{\psi_1, \ldots, \psi_m\}$, and $cons(\delta) = \theta$;
- $\Pi = (\delta_0, \delta_1, \ldots)$ is a finite or infinite sequence of defaults from $\mathcal{D}$ without multiple occurrences; $\Pi[k]$ denotes the initial segment of Π of length k;
- $In(\Pi)$ is the deductive closure of $(\mathcal{W} \cup \{cons(\delta) \mid \delta \text{ occurs in } \Pi\})$;
- $Out(\Pi) = \{\neg\psi \mid \psi \in just(\delta) \text{ for some } \delta \text{ occurring in } \Pi\}$;
- Π is called a *process of* $\mathcal{T}$ iff δ_k is applicable to $In(\Pi[k])$ whenever δ_k occurs in Π;
- Π is *successful* iff $In(\Pi) \cap Out(\Pi) = \emptyset$, otherwise it is *failed*;
- Π is *closed* iff every $\delta \in \mathcal{D}$ that is applicable to $In(\Pi)$ already occurs in Π.

Now we can give the definition of extension, according to [2]: a set of formulae E is an *extension* of the default theory $\mathcal{T}$ iff there is some closed and successful process Π of $\mathcal{T}$ such that $E = In(\Pi)$.

6.2 Lifting dl-programs to add default rules

Now we show how to add default rules to dl-programs.

In this setting, we only consider semi-normal default rules, i.e. rules $\frac{\varphi_1, \ldots, \varphi_n : \psi_1, \ldots, \psi_m}{\theta}$ where θ is ψ_k for some $k = 1, \ldots, m$; we also assume that φ_i and ψ_j are all atomic formulas or their negations, that is, each φ_i is $A_i(\overline{y_i})$ or $\neg A_i(\overline{y_i})$ for some predicate A_i and each ψ_j is $B_j(\overline{z_j})$ or $\neg B_j(\overline{z_j})$ for some predicate B_j.

As Motik argues, we are interested mostly in cases where $A_1, \ldots, A_n, B_1, \ldots, B_m$ originate from $\mathcal{L}$, since this is the situation that cannot be dealt with directly in dl-programs. The construction we introduce below can be easily adapted to the case where some of the A_i come from $\mathcal{P}$; as we will discuss below, the case where B_j comes from $\mathcal{P}$ is not very interesting.

Let $\mathcal{KB}_\Gamma$ be a dl-program with lifting, where $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ is a dl-program and Γ may be empty, to which we want to add a semi-normal, negation-free default rule of the following form.

$$\frac{A_1(\overline{y_1}), \ldots, A_n(\overline{y_n}) : B_1(\overline{z_1}), \ldots, B_m(\overline{z_m})}{B_k(\overline{z_k})}$$

As a first step, we add B_k to Γ. Note that Γ may also contain some of the A_i or B_j if they happen to have been lifted before.

At this stage, the reason for extending lifted predicates (in particular B_k) in all queries in $\mathcal{KB}_\Gamma$ should also be clear, since there is no other way of refeeding the new conclusions into the knowledge base $\mathcal{L}$.

We can now take advantage of the interplay between B_k and its lifting b_k^+, and model the above rule as follows.

$$b_k^+(\overline{z_k}) \leftarrow DL_\Gamma[; A_1](\overline{y_1}), \ldots, DL_\Gamma[; A_n](\overline{y_n}), \text{not } DL_\Gamma[; \neg B_1](\overline{z_1}), \ldots, \text{not } DL_\Gamma[; \neg B_m](\overline{z_m}) \quad (6)$$

If the rule contains negated atoms, the changes in the dl-queries are straightforward; if we want to have $\neg B_k(\overline{z_k})$ as the conclusion, then the head of the encoding should be $b_k^-(\overline{z_k})$. For simplicity of notation, we will write default rules in our results as if they did not contain negation; the proofs are straightforward to adapt to the general case.

Note that this construction is very similar to the Transformation Ω of default theories to cq-programs (a variant of dl-programs) introduced in [13]. The main difference is our usage of the lifting construction presented earlier, which the authors of that paper do not consider (at least explicitly). A deeper comparison of both approaches can be found in Section 8.3.

Once again, we have a useful consistency preservation result that holds as long as we are working with a complete description logic. This is not a serious restriction, since many useful description logics are complete, anyway.

Theorem 2. Let $\mathcal{KB}_\Gamma = \langle \mathcal{L}, \mathcal{P}_\Gamma \rangle$ be a dl-program with lifting where the description logic underlying $\mathcal{L}$ is complete, and $\mathcal{KB}^+_{\Gamma^+}$ the dl-program with lifting where $\Gamma^+ = \Gamma \cup \{B_k\}$ and $\mathcal{P}^+$ is $\mathcal{P}$ together with the encoding of the semi-normal default rule $\frac{A_1, \ldots, A_n : B_1, \ldots, B_m}{B_k}$ as in (6). If either $B_k \in \Gamma$ or b_k^+ and b_k^- are not used in the head of any rule in $\mathcal{P}$ then for every dl-atom in $\mathcal{P}$, if the knowledge base underlying its query in $\mathcal{KB}_\Gamma$ is consistent, then so is the knowledge base underlying it in $\mathcal{KB}^+_{\Gamma^+}$.

Proof. We proceed by contradiction. Suppose that there is a dl-atom $DL_{\Gamma^+}[\chi; R](t)$ in $\mathcal{P}_{\Gamma^+}$ whose underlying knowledge base is inconsistent. Let $\mathcal{L}^*$ be the knowledge base underlying this query in $\mathcal{KB}_\Gamma$.

If $B_k \in \Gamma$, then $\mathcal{P}_\Gamma$ and $\mathcal{P}_{\Gamma^+}$ only differ in the presence of rule (6), so the inconsistency must come from enriching $\mathcal{L}^*$ with $B_k(t)$ for some finite set of terms $t_1, \ldots, t_p$ such that $b_k^+(t_i)$ were obtained from this rule. From the well-founded semantics for dl-programs, this means in particular that $DL_\Gamma[; \neg B_k](t_i)$ cannot be proved regardless of how the extent of b_k^+ is increased. By completeness of the description logic, this implies that $\mathcal{L}^* \cup \{B_k(t_i)\}$ is consistent. Iterating this reasoning for $i = 1, \ldots, p$, the thesis follows.

If $B_k \notin \Gamma$, then by Theorem 1 the dl-program $\mathcal{KB}_{\Gamma^+}$ (where B_k has been added to Γ but $\mathcal{P}$ has not been changed) still satisfies the hypothesis, and is in the condition of the previous paragraph. □

If some A_i comes from $\mathcal{P}$ and not $\mathcal{L}$, the corresponding dl-atom $DL_\Gamma[; A_i](\overline{y_i})$ is simply replaced with $A_i(\overline{y_i})$. The case where B_j comes from $\mathcal{P}$ presents some specific technical problems, and will not be dealt with here.

As it turns out, we can actually prove stronger results partially characterising the semantics of dl-programs with this encoding of default rules.

We start by giving two examples showing that our definitions correspond to some intuitive notion of default rule.

Example 4. Consider a scenario where we want to reason about people who may be students or teachers, and a person is assumed to be a student unless there is contradicting evidence. This can be modeled by means of the default rule

$$\frac{\mathsf{person}(x) : \mathsf{student}(x), \neg\mathsf{teacher}(x)}{\mathsf{student}(x)}$$

Furthermore, assume the knowledge base contains the single fact $\mathsf{person}(\mathsf{john})$. Since nothing is known about john except for his being a person, the standard semantics for default logics yields a single extension containing the two facts $\mathsf{person}(\mathsf{john})$ and $\mathsf{student}(\mathsf{john})$.

Translating this to a dl-program, we take $\Gamma = \{\mathsf{student}\}$ and obtain the logic program

$$\begin{aligned}
&\mathsf{student}^+(X) \leftarrow DL_\Gamma[;\mathsf{person}](X), \text{not } DL_\Gamma[;\neg\mathsf{student}](X), \text{not } DL_\Gamma[;\mathsf{teacher}](X)\\
&\mathsf{student}^+(X) \leftarrow DL_\Gamma[;\mathsf{student}](X)\\
&\mathsf{student}^-(X) \leftarrow DL_\Gamma[;\neg\mathsf{student}](X)
\end{aligned}$$

We now compute the well-founded semantics of this program, as described in Section 2.1. The details are left to the reader.

$$\begin{aligned}
I_0 &= \emptyset & I_1 &= \{\neg\mathsf{student}^-(\mathsf{john})\}\\
T_{\mathcal{KB}_\Gamma}(I_0) &= \emptyset & T_{\mathcal{KB}_\Gamma}(I_1) &= \{\mathsf{student}^+(\mathsf{john})\}\\
U_{\mathcal{KB}_\Gamma}(I_0) &= \{\mathsf{student}^-(\mathsf{john})\} & U_{\mathcal{KB}_\Gamma}(I_1) &= \{\mathsf{student}^-(\mathsf{john})\}\\
W_{\mathcal{KB}_\Gamma}(I_0) &= \{\neg\mathsf{student}^-(\mathsf{john})\} & W_{\mathcal{KB}_\Gamma}(I_1) &= \{\mathsf{student}^+(\mathsf{john}), \neg\mathsf{student}^-(\mathsf{john})\}
\end{aligned}$$

Since $W_{\mathcal{KB}_\Gamma}(I_1)$ cannot be consistently extended, it is the lfp of $W_{\mathcal{KB}_\Gamma}$, and hence coincides with the well-founded semantics of the program. This program proves $\mathsf{person}(\mathsf{john})$ (in the description logic) and $\mathsf{student}^+(\mathsf{john})$ (in the logic program part), coinciding with the Reiter semantics discussed above.

The other fact contained in the well-founded semantics is $\neg\mathsf{student}^-(\mathsf{john})$. This fact states that $\mathsf{student}^-(\mathsf{john})$ is unfounded, in other words, one will never be able to prove that john is *not* a student.

However, when the default rules produce multiple extensions, the semantics of the dl-program cannot coincide, since it only contains one model. The following example shows that this model also makes sense, intuitively.

Example 5. Suppose we extend the previous scenario with a default rule stating that a person is assumed to be a teacher unless there is contradicting evidence. We now have the two default rules

$$\frac{\mathsf{person}(x) : \mathsf{student}(x), \neg\mathsf{teacher}(x)}{\mathsf{student}(x)} \qquad \frac{\mathsf{person}(x) : \mathsf{teacher}(x), \neg\mathsf{student}(x)}{\mathsf{teacher}(x)}$$

with the knowledge base as before.

Since nothing is known about john except for his being a person, either rule can be applied to infer that john is a student or that john is a teacher – but not both simultaneously, since the two rules mutually block each other. Hence, in the standard semantics for default logics there are two different extensions, namely $\{\mathsf{person}(\mathsf{john}), \mathsf{student}(\mathsf{john})\}$ and $\{\mathsf{person}(\mathsf{john}), \mathsf{teacher}(\mathsf{john})\}$.

Translating this to a dl-program, we now take $\Gamma = \{\mathsf{student}, \mathsf{teacher}\}$ and obtain the following logic program.

$$\begin{aligned}
&\mathsf{student}^+(X) \leftarrow DL_\Gamma[;\mathsf{person}](X), \text{not } DL_\Gamma[;\neg\mathsf{student}](X), \text{not } DL_\Gamma[;\mathsf{teacher}](X)\\
&\mathsf{teacher}^+(X) \leftarrow DL_\Gamma[;\mathsf{person}](X), \text{not } DL_\Gamma[;\neg\mathsf{teacher}](X), \text{not } DL_\Gamma[;\mathsf{student}](X)
\end{aligned}$$

$$\mathsf{student}^+(X) \leftarrow DL_\Gamma[;\mathsf{student}](X) \qquad \mathsf{teacher}^+(X) \leftarrow DL_\Gamma[;\mathsf{teacher}](X)$$
$$\mathsf{student}^-(X) \leftarrow DL_\Gamma[;\neg\mathsf{student}](X) \qquad \mathsf{teacher}^-(X) \leftarrow DL_\Gamma[;\neg\mathsf{teacher}](X)$$

We compute the well-founded semantics of this program as before.

$$I_0 = \emptyset \qquad I_1 = \{\neg\mathsf{student}^-(\mathsf{john}), \neg\mathsf{teacher}^-(\mathsf{john})\}$$
$$T_{\mathcal{KB}_\Gamma}(I_0) = \emptyset \qquad T_{\mathcal{KB}_\Gamma}(I_1) = \emptyset$$
$$U_{\mathcal{KB}_\Gamma}(I_0) = \{\mathsf{student}^-(\mathsf{john}), \mathsf{teacher}^-(\mathsf{john})\} \qquad U_{\mathcal{KB}_\Gamma}(I_1) = U_{\mathcal{KB}_\Gamma}(I_0)$$
$$W_{\mathcal{KB}_\Gamma}(I_0) = \{\neg\mathsf{student}^-(\mathsf{john}), \neg\mathsf{teacher}^-(\mathsf{john})\} \qquad W_{\mathcal{KB}_\Gamma}(I_1) = I_1$$

Hence the dl-program only proves $\mathsf{person}(\mathsf{john})$. This makes sense: it is the only fact that can *always* be proved, i.e. it belongs to the intersection of both Reiter extensions.

Note that the well-founded semantics also informs us that we cannot *refute* either $\mathsf{student}(\mathsf{john})$ or $\mathsf{teacher}(\mathsf{john})$. In other words, there may be (and in this case there are) extensions in which each of these facts holds.

This example suggests that our encoding of default rules is related to the skeptic approach to the extension semantics of default logics: we only accept facts that hold in all extensions. We may even infer facts that would be rejected by the safe approach, i.e. facts that hold in all extensions but not for the same reason, as the following example shows.

Example 6. Consider the following two default rules together with an empty knowledge base.

$$\frac{:P,Q}{Q} \qquad \frac{:\neg P,Q}{Q}$$

These rules generate the following dl-program, where $\Gamma = \{Q\}$.

$$q^+ \leftarrow \text{not } DL_\Gamma[;\neg P], \text{not } DL_\Gamma[;\neg Q] \qquad q^+ \leftarrow DL_\Gamma[;Q]$$
$$q^+ \leftarrow \text{not } DL_\Gamma[;P], \text{not } DL_\Gamma[;\neg Q] \qquad q^- \leftarrow DL_\Gamma[;\neg Q]$$

The well-founded semantics is simple to compute:

$$I_0 = \emptyset \qquad I_1 = \{\neg q^-\}$$
$$T_{\mathcal{KB}_\Gamma}(I_0) = \emptyset \qquad T_{\mathcal{KB}_\Gamma}(I_1) = \{q^+\}$$
$$U_{\mathcal{KB}_\Gamma}(I_0) = \{q^-\} \qquad U_{\mathcal{KB}_\Gamma}(I_1) = U_{\mathcal{KB}_\Gamma}(I_0)$$
$$W_{\mathcal{KB}_\Gamma}(I_0) = \{\neg q^-\} \qquad W_{\mathcal{KB}_\Gamma}(I_1) = \{q^+, \neg q^-\}$$

and $W_{\mathcal{KB}_\Gamma}(I_1)$ is the well-founded semantics of this program. Note, however, that Q is a skeptic consequence of these default rules but not a safe one, since the process tree for them has two leaves yielding two extensions that contain Q for different reasons (namely, one corresponding to each default rule).

The following theorem formalizes the connection between the process tree semantics for default rules – the semantics that has been used in all previous examples – and the induced semantics from the well-founded semantics for dl-programs, while simultaneously clarifying the meaning of unfounded atoms. For simplicity of notation, we identify each predicate P with its lifting p^+ and its negation $\neg P$ with p^-.

Theorem 3. Let $\mathcal{KB}_\Gamma$ be a dl-program including the encoding of a set $\mathcal{D}$ of default rules. Let $\{E_j\}_{j\in J}$ be the set of extensions of $\mathcal{D}$ and $W = WFS(\mathcal{KB}_\Gamma)$ be the well-founded semantics of $\mathcal{KB}_\Gamma$, with W^+ the set of atoms in W and W^- the set of atoms whose negations occur in W.
Then $W^+ \subseteq \bigcap_{j\in J} E_j$ and $W^- \cap \bigcup_{j\in J} E_j = \emptyset$.

This is a consequence of the following lemma.

Lemma 2. Let $\mathcal{KB}_\Gamma$ and $\mathcal{D}$ be as before, and let Π be a closed and successful process of $\mathcal{D}$ (in other words, $In(\Pi)$ corresponds to an extension of $\mathcal{D}$). If $J^+ \subseteq In(\Pi)$ and $U \cap In(\Pi) = \emptyset$, then:

1. $T_{\mathcal{KB}_\Gamma}(J \cup \neg U) \subseteq In(\Pi)$;
2. $U_{\mathcal{KB}_\Gamma}(J \cup \neg U) \cap In(\Pi) = \emptyset$.

Proof.

1. There are two cases to considered.

   If $q^+(\bar{t})$ is included in $T_{\mathcal{KB}_\Gamma}(J \cup \neg U)$ because of the encoding of a closed instance δ

   $$q^+\left(\overline{t}\right) \leftarrow DL_\Gamma[;A_1]\left(\overline{t_1}\right), \ldots, DL_\Gamma[;A_n]\left(\overline{t_n}\right), \text{not } DL_\Gamma[;\neg B_1]\left(\overline{s_1}\right), \ldots, \text{not } DL_\Gamma[;\neg B_m]\left(\overline{s_m}\right) .$$

   of a default rule in $\mathcal{D}$, then $J^+ \models DL_\Gamma[;A_i]\left(\overline{t_i}\right)$ for each i, hence $In(\Pi) \models A_i\left(\overline{t_i}\right)$ by hypothesis. Also, for each consistent $S \supseteq (J \cup \neg U)$, $S^+ \not\models DL_\Gamma[;\neg B_i]\left(\overline{s_i}\right)$ for each i. Since $U \cap In(\Pi) = \emptyset$, the set $\neg U \cup In(\Pi)$ is consistent; also, it expands $J \cup \neg U$, and therefore $\neg U \cup In(\Pi) \not\models \neg B_i\left(\overline{s_i}\right)$, hence $In(\Pi) \not\models \neg B_i\left(\overline{s_i}\right)$.

   We conclude that δ is applicable to $In(\Pi)$. But Π is closed, so δ occurs in Π and therefore $Q(t) \in In(\Pi)$. Since this holds for every default rule extending $T_{\mathcal{KB}_\Gamma}(J \cup \neg U)$, we conclude that $T_{\mathcal{KB}_\Gamma}(J \cup \neg U) \subseteq In(\Pi)$.

   If $q^+(t)$ is included in $T_{\mathcal{KB}_\Gamma}(J \cup \neg U)$ because of some other (non-default) rule, then $q^+(t) \in In(\Pi)$ by definition of $In(\Pi)$.

2. Assume now that $q^+(t) \in U_{\mathcal{KB}_\Gamma}(J \cup \neg U)$, and let δ be the encoding of a closed instance

   $$q^+\left(\overline{t}\right) \leftarrow DL_\Gamma[;A_1]\left(\overline{t_1}\right), \ldots, DL_\Gamma[;A_n]\left(\overline{t_n}\right), \text{not } DL_\Gamma[;\neg B_1]\left(\overline{s_1}\right), \ldots, \text{not } DL_\Gamma[;\neg B_m]\left(\overline{s_m}\right) .$$

   of a default rule in $\mathcal{D}$.

   There are two possibilities. Suppose that $J^+ \models DL_\Gamma[;\neg B_i]\left(\overline{s_i}\right)$ for some i; then $In(\Pi) \models \neg B_i\left(\overline{s_1}\right)$ by hypothesis. On the other hand, if for some $S \supseteq J$ we have $S^+ \not\models DL_\Gamma[;A_i]\left(\overline{t_i}\right)$ for some i, then reasoning as before we conclude that $In(\Pi) \not\models A_i\left(\overline{t_i}\right)$. In either case, δ is not applicable to Π. Since this holds for every rule having $Q(t)$ as its conclusion, we conclude that $Q(t) \notin In(\Pi)$, whence $U_{\mathcal{KB}_\Gamma}(J \cup \neg U) \cap In(\Pi) = \emptyset$.

□

Proof (of Theorem 3). We can compute the well-founded semantics of a dl-program $\mathcal{KB}_\Gamma = \langle \mathcal{L}, \mathcal{P}_\Gamma \rangle$ by means of the following three sequences of sets.

$$\begin{array}{ccc} J_0 = \emptyset & U_0 = \emptyset & I_0 = J_0 \cup \neg U_0 \\ J_{k+1} = T_{\mathcal{KB}_\Gamma}(I_k) & U_{k+1} = U_{\mathcal{KB}_\Gamma}(I_k) & I_{k+1} = J_{k+1} \cup \neg U_{k+1} \end{array}$$

All three sequences eventually reach a fixed-point. Let n be the index of such a fixed-point; it is easy to see that $I_n = WFS(\mathcal{KB}_\Gamma) = W$. Because of how I_n is defined, it follows that $J_n = W^+$ and $U_n = W^-$.

Trivially, $J_0 \subseteq In(\Pi)$ and $U_0 \cap In(\Pi) = \emptyset$ for any closed and successful process Π. By induction using the previous lemma for the induction step, it follows that $J_k \subseteq In(\Pi)$ and $U_k \cap In(\Pi) = \emptyset$ for every k. In particular, $J_n \subseteq In(\Pi)$ and $U_n \cap In(\Pi) = \emptyset$, thus establishing the desired result. □

Although the examples above show that this formalization of default rules has some similarities with the skeptic approach to default reasoning, the sets W^+ and W^- are not necessarily the largest sets satisfying the conditions of Theorem 3. We illustrate this with two (slightly pathological) examples.

The first example shows that there may be skeptical consequences of a set of default rules that are not included in the well-founded semantics of the corresponding dl-program.

Example 7. Consider the following set of default rules.

$$\frac{P : Q}{Q} \qquad \frac{\neg P : Q}{Q} \qquad \frac{: P}{P} \qquad \frac{: \neg P}{: \neg P}$$

This set of default rules admits two extensions: $E_1 = \{P, Q\}$ and $E_2 = \{\neg P, Q\}$. The skeptic approach will therefore admit Q as a consequence of these rules.

The corresponding dl-program is the following, with $\Gamma = \{P, Q\}$.

$$\begin{array}{ll} q^+ \leftarrow DL_\Gamma[; P], \text{not } DL_\Gamma[; \neg Q] & p^+ \leftarrow DL_\Gamma[; P] \\ q^+ \leftarrow DL_\Gamma[; \neg P], \text{not } DL_\Gamma[; \neg Q] & p^- \leftarrow DL_\Gamma[; \neg P] \\ p^+ \leftarrow \text{not } DL_\Gamma[; \neg P] & q^+ \leftarrow DL_\Gamma[; Q] \\ p^- \leftarrow \text{not } DL_\Gamma[; P] & q^- \leftarrow DL_\Gamma[; \neg Q] \end{array}$$

However, the well-founded semantics is more meager than expected.

$$\begin{array}{rr} I_0 = \emptyset & I_1 = \{\neg q^-\} \\ T_{\mathcal{KB}_\Gamma}(I_0) = \emptyset & T_{\mathcal{KB}_\Gamma}(I_1) = \emptyset \\ U_{\mathcal{KB}_\Gamma}(I_0) = \{q^-\} & U_{\mathcal{KB}_\Gamma}(I_1) = U_{\mathcal{KB}_\Gamma}(I_0) \\ W_{\mathcal{KB}_\Gamma}(I_0) = \{\neg q^-\} & W_{\mathcal{KB}_\Gamma}(I_1) = \{\neg q^-\} = WFS(\mathcal{KB}_\Gamma) \end{array}$$

The problem is that nothing absolute is ever deduced about P, so none of the default rules is applicable to yield q^+. Hence in this case $WFS(\mathcal{KB}_\Gamma)^+ \subsetneq E_1 \cap E_2$.

Note that Q is *not* a safe consequence of the set of default rules in this last example. It is reasonable to expect that all safe consequences are included in the well-founded semantics for dl-programs, which would place the latter strictly between safe and skeptic reasoning. However, no rigorous proof of this fact is known to the authors.

The other inclusion in Theorem 3 may also fail to hold, as the following example shows.

Example 8. Consider the following set of default rules.

$$\frac{: P, \neg Q}{P} \qquad \frac{: Q, \neg P}{Q} \qquad \frac{: \neg P, \neg Q, R}{R}$$

From this rules we obtain two different extensions: $\{P\}$ and $\{Q\}$. The first two rules are mutually blocking, while the third rule is not applicable in any extension. Furthermore, if one attempts to build an extension by starting with the third rule, one quickly runs into problems, as this does not block either of the other rules but the results are incompatible.

The corresponding dl-program has $\Gamma = \{P, Q, R\}$ and contains the following rules.

$$\begin{array}{lll} p^+ \leftarrow \text{not } DL_\Gamma[; \neg P], \text{not } DL_\Gamma[; Q] & p^+ \leftarrow DL_\Gamma[; P] & p^- \leftarrow DL_\Gamma[; \neg P] \\ q^+ \leftarrow \text{not } DL_\Gamma[; \neg Q], \text{not } DL_\Gamma[; P] & q^+ \leftarrow DL_\Gamma[; Q] & q^- \leftarrow DL_\Gamma[; \neg Q] \\ r^+ \leftarrow \text{not } DL_\Gamma[; P], \text{not } DL_\Gamma[; Q], \text{not } DL_\Gamma[; \neg R] & r^+ \leftarrow DL_\Gamma[; R] & r^- \leftarrow DL_\Gamma[; \neg R] \end{array}$$

The well-founded semantics is again very simple to compute.

$$\begin{array}{rr} I_0 = \emptyset & I_1 = \{\neg p^-, \neg q^-, \neg r^-\} \\ T_{\mathcal{KB}_\Gamma}(I_0) = \emptyset & T_{\mathcal{KB}_\Gamma}(I_1) = \emptyset \\ U_{\mathcal{KB}_\Gamma}(I_0) = \{p^-, q^-, r^-\} & U_{\mathcal{KB}_\Gamma}(I_1) = U_{\mathcal{KB}_\Gamma}(I_0) \\ W_{\mathcal{KB}_\Gamma}(I_0) = \{\neg p^-, \neg q^-, \neg r^-\} & W_{\mathcal{KB}_\Gamma}(I_1) = \{\neg p^-, \neg q^-, \neg r^-\} = WFS(\mathcal{KB}_\Gamma) \end{array}$$

We would hope to have r^+ in the set of unfounded atoms, since there is no way to prove it. However, the semantics for dl-programs do not allow this, since it is always possible to extend the empty set in such a way that $DL_\Gamma[; P]$ or $DL_\Gamma[; Q]$ holds, which prevents us from adding r^+ to $U_{\mathcal{KB}_\Gamma}(\emptyset)$. The problem is that R is not derivable in any extension, but the *reason* for this (i.e. the premise in the default rule that is violated) is different in each extension.

We close this section with two important remarks.

First, Theorem 3 compares the well-founded semantics with the intersection and the union of all Reiter-style extensions of the set of default rules. If there are no extensions (which can happen, as is well known), the theorem simply states that W^+ and W^- are sets of atoms, which is not very informative. Still, in such a situation our dl-programs *do* allow some inferences. Consider the three default rules

$$\frac{:P,\neg Q}{P} \qquad \frac{:Q,\neg R}{Q} \qquad \frac{:R,\neg P}{R}$$

and encode them in a dl-program together with the single fact $A \leftarrow$. The corresponding default logic is well-known to have no extensions, but the dl-program will still prove A (and, incidentally, p^-, q^- and r^- will all be unfounded atoms). One may think these default rules are being ignored, but they do affect the set of consequences: the single rule $\frac{:\neg P}{\neg P}$ allows for the single extension $\{\neg P\}$, which is also deduced by the corresponding dl-program, but when we couple it with the three default rules above there are no extensions, and the generated dl-program also proves no positive facts – so the presence of those three default rules is effectively preventing us from deducing $\neg P$.

The second remark is of a positive nature. Although Theorem 3 only guarantees inclusions, in many concrete examples those inclusions are in fact equalities. A simple case (which is straightforward to check) happens when $W^+ \cup W^-$ contains all closed atomic formulas in the knowledge base. Another interesting case is when $W^+ \cup W^-$ contains all closed instances of q^+ and q^- (simultaneously), in which case the inclusions in the theorem become equalities for instances of Q. These properties are all consequences of the consistency of the dl-program's extended knowledge base (Theorem 2).

7 Discussion

We will now show how our constructions solve the problems detected in Section 3 and discuss our contributions to this area of research.

The example displayed in Table 3 presented two problematic issues, concerning two distinct situations, which we will now show can be solved using the techniques introduced in the intervening sections.

Consider first rule (9), which reads

$$\mathbf{K}\ \mathsf{seasideCity}(x) \leftarrow \mathbf{K}\ \mathsf{portCity}(x), \mathbf{not}\ \neg\mathsf{seasideCity}(x)$$

This rule was marked as $(*)$ in the translation, since it could not be directly translated in a dl-program $\mathcal{KB} = \langle \mathcal{L}, \mathcal{P} \rangle$ setting because of the mixture between the two components: the rule deals with predicates from $\mathcal{L}$, but the head cannot contain a dl-atom. Noting that it is a default rule, we can now translate it (indirectly) by lifting $\mathsf{seasideCity}$ to $\mathcal{P}$ and encoding the rule as discussed in Section 6, obtaining

$$\begin{aligned}
\mathsf{seasideCity}^+(X) &\leftarrow DL_\Gamma[;\mathsf{portCity}](X), \mathrm{not}\ DL_\Gamma[;\neg\mathsf{seasideCity}](X)\\
\mathsf{seasideCity}^+(X) &\leftarrow DL_\Gamma[;\mathsf{seasideCity}](X)\\
\mathsf{seasideCity}^-(X) &\leftarrow DL_\Gamma[;\neg\mathsf{seasideCity}](X)
\end{aligned}$$

where $\Gamma = \{\mathsf{seasideCity}\}$. Observe that each dl-atom in every other rule in the translation (namely in Rules (10) and (13)) is affected by this lifting, as was discussed earlier.

Rules (11) and (12) dealt with an integrity constraint very similar to the examples in Section 5. The problem here was having false as head of a rule; as we discussed earlier, there is even no reason why this should be desirable. Adding a fresh predicate $\bot_{\mathsf{hasOnSea}}$ we can translate these rules almost directly.

$$\begin{aligned}
\mathsf{hasOnSea}(x) &\leftarrow DL_\Gamma[;\mathsf{onSea}](x,y)\\
\bot_{\mathsf{hasOnSea}} &\leftarrow DL_\Gamma[;\mathsf{seasideCity}](x), \mathrm{not}\ \mathsf{hasOnSea}(x)
\end{aligned}$$

Note that the dl-atoms are extended due to the lifting that was done earlier. The user can check that the knowledge base $\mathcal{P}$ is consistent at any given time by calling the goal $\bot_{\mathsf{hasOnSea}}$. Also observe that this is all we can do: in this concrete example, if the integrity constraint is not satisfied, we cannot complete the knowledge base because that would require guessing a value for the sea bathing a particular city.

The methodologies we developed in the previous sections make dl-programs able to represent the example from [25]. Furthermore, the resulting dl-program is much cleaner than the original MKNF formalization presented in that paper, since the predicates from the knowledge base and the logic program are kept separate at all times, with interaction between them kept to a minimum and always explicitly mentioned. This has a practical interest for a number of reasons.

The two components of a dl-program are typically designed semi-independently, with the knowledge base usually being available somewhere; our approach allows a programmer to use a knowledge base (even enriching it with default rules) without having to change it, whereas an MKNF-style encoding of defaults quickly erases the distinction between the original knowledge base and the final result. Also, knowledge bases are often meant to be reutilized, and extensions with e.g. default rules may be specific to some context. Our approach allows local extensions, once again without requiring any change to the original knowledge base – meaning it can be independently upgraded or reused regardless of the dl-program(s) it is being used in.

The lifting technique, which is effectively new although its key ingredients are already present in dl-programs, is also a powerful tool, since it allows predicates to be completely shared between the two components of a dl-program (and in a global sense) without mixing them as in MKNF. The theoretical results we proved show that lifting a predicate P to p^+ and p^- creates two new predicates intimately connected with the original one: p^+ coincides with P, p^- coincides with $\neg P$. Once established, this connection is permanent and dynamic: any changes made to P will be reflected in p^+ and p^-, and vice-versa. Also, the consistency preservation results guarantee that the consistency of the description logic is not affected by lifting (although, of course, it will be affected by any subsequent reckless usage of p^+ or p^-).

As a side effect, lifting also solves one of the main shortcomings of dl-programs pointed in [25]: its lack of tightness. The interaction achived by lifting makes dl-programs tight in a controlled way; furthermore, as long as information about which predicates have been lifted is dynamically maintained together with the program, the updates to dl-queries can be made automatically, so that the user can in practice forget about keeping track of the set Γ being appended to dl-atoms.

Our treatment of default rules is very similar to Motik's proposal, and corresponds to translating his approach directly to dl-programs with lifting. However, from [25] one is not able to understand exactly what is being captured by this translation of default rules. Clearly their semantics must differ from standard default semantics (since these all rely on the existence of different, often contradictory, extensions, while MKNF semantics, like the well-founded semantics of dl-programs, only produces one model), but the authors completely avoid the discussion of *what* semantics they are obtaining. Section 6 gives a partial characterization of our encoding of default rules in dl-programs and shows that we can prove some, but not all, of the facts that hold in all Reiter-style extensions, and that we obtain as unfounded some, but not all, of the facts that are false in all Reiter-style extensions. These inclusions have been proven formally, and we provided counter-examples showing that they are strict, together with a criterion that can be used to show equality in specific situations. We also conjecture that all safe consequences of a set of default rules are provable in the dl-program implementation. Furthermore, our counter-examples all rely on the presence of rules with negative formulas in their conclusion, leaving the open question of whether the stated inclusions become equalities if only positive conclusions are allowed in default rules.

The possibility of implementing default rules extending predicates that are originally defined in the knowledge base is an aspect of the flexibility of the system. Our approach shows that dl-programs can be made much more flexible than one might suppose at first glance, since lifting (open-world) predicates from the knowledge base to the logic program allows them to be treated as closed-world entities. Although this capability is already present in [15], our examples illustrate how this is put to work through the interaction of lifting and default rules. The approach of [16, 13] also explores this capability, but the authors never isolated the lifting construction. As a result, predicates extended by default rules become split between the knowledge base and the logic program, making the semantics of the resulting program trickier to understand. This issue is examined in more detail in Section 8.3.

All our work is done entirely in the setting of the syntactic constructions already available within the framework of dl-programs. This means that all complexity results which have been proved elsewhere (see e.g. [16, 15]) hold for dl-programs including liftings, integrity constraints or default rules implemented as we describe. Note, however, that we make some hefty demands on the expressiveness of the underlying description logics (they must have negation, for starters) – which means, in particular, that we will not

be able to get nice complexity results as when working with Datalog-rewritable logics [19].

8 Related work

Many other authors have worked on bringing together the world of description logics and several other features that we discuss here, namely default rules and integrity constraints. In this section we summarize some of this work, most of which is not as directly related to our contribution as the articles discussed in more detail in earlier sections.

8.1 Extending Datalog to express integrity constraints

An approach quite different from ours consists in the proposal of a family of extensions of Datalog by integrity constraints such as *tgds*, called $Datalog^{\pm}$[8], allowing existentially quantified variables or the truth constant *false*, in the head of rules. Several sublanguages of $Datalog^{\pm}$ are defined for which query answering is decidable and complexity results are given. For a variable set of *tgds*, the program complexity of linear $Datalog^{\pm}$ (only allowing rules with one atom in the body and one atom in the head) is PSPACE-complete, and for guarded $Datalog^{\pm}$ (rules having a guard in the body, i.e. an atom that contains all the universally quantified variables in the rule) it is 2EXPTIME-complete.

$Datalog^{\pm}$ is further extended by negative constraints and *egds*. However, to avoid the undecidability of query answering that can result from the interaction of *tgds* end *egds* even in simple cases [9], these authors consider only a restricted class of *egds*, namely key dependencies (functional dependencies expressing a key k of a relation r, which are a special case of *egds*) that are non-conflicting with the set T of *tgds* in the program (a key k of a relation r is non-conflicting with a tgd t, if r is different from the ψ predicate in the head of t , or the positions of k in r are not a proper subset of the $\vec{x}$ positions in ψ).

$Datalog^{\pm}$ is also further extended with stratified negation: *tgd* rules are extended to allow (true) negation in the rule bodies, conjunctive queries are also extended to allow negated atoms, and a perfect model semantics of guarded $Datalog^{\pm}$ with stratified negation is defined.

8.2 Extending OWL to express integrity constraints

In [29, 30] OWL ontologies are augmented with integrity constraint axioms. While the standard OWL open world semantics is used for standard OWL axioms, integrity constraint axioms are used to validate instance data using closed world semantics and a weak form of the unique names assumption (UNA).

In this weak form of UNA, any named individuals with different identifiers are assumed to be different by default, unless their equality is required to satisfy the axioms in the OWL knowledge base. In the integrity constraints semantics, any named individual that cannot be proven to be an instance of a concept C is assumed to be an instance of $\neg C$.

In order to check the validity of an extended knowledge base $\langle \mathcal{K}, \mathcal{C} \rangle$ where $\mathcal{C}$ is a set of integrity constraints, [29, 30] define translation rules from integrity axioms to DCQ^{not} – distinguished conjunctive queries (conjunctive queries using only variables that can only be mapped to known individuals) using `not` (negation as failure). To be able to prove that integrity constraints validation can be reduced to DCQ^{not} query answering, they are forced to reduce the expressivity of the description logic language to $\mathcal{SRI}$ or the integrity constraints to ones that do not contain cardinality restrictions. As SPARQL can express DCQ^{not}, a further translation to SPARQL allows any OWL reasoner that supports SPARQL query answering to be used for integrity constraints validation.

Another approach that uses OWL as an integrity constraint language has been proposed in [23, 24]. In this work, a description logic knowledge base is extended with a set $\mathcal{C}$ of constraint TBox axioms. Constraint TBox axioms are checked only w.r.t. the minimal Herbrand models of the union of the ABox and the (initial, not extended) TBox. However, [29, 30] argue that this approach can give results that are not desirable for closed world data validation, namely because unnamed individuals (resulting from existential quantifiers in the axioms) can satisfy constraints.

8.3 Other approaches on extending DL with default rules

The problem of extending description logics with default rules has been extensively studied in the past twenty years, and it is impossible to mention all the relevant results obtained by other authors. In this

section, we focus on work that is most closely related to the construction we introduced: the approach proposed by Motik et al. [25], modeling default rules in MKNF, and the work of Eiter et al. [16] and Dao-Tran et al. [13].

As the example from Section 3 shows, default rules are reasonably straightforward to implement in MKNF knowledge bases, yielding a formalism equivalent to that of Baader and Hollunder [5]. Unfortunately, Motik's approach capitalizes on the lack of separation between the knowledge bases and reasoning components of MKNF knowledge bases. The dl-programs cleanly separate these two components, which we feel to be a desirable property.

In contrast, [16] and [13] implement default rules in dl-programs (the latter actually considers cq-programs, which are a variant of dl-programs where dl-atoms are allowed to have disjunctive queries and heads of rules are allowed to be disjunctions of literals). The original work presents a sophisticated construction that was later simplified, yielding encodings of default rules that look very much like our proposal. For example, the encoding of Example 4 using Transformation Ω would be as follows.

$$\mathsf{in}_{\mathsf{student}}(X) \leftarrow DL[\lambda; \mathsf{person}](X), \mathrm{not}\, DL[\lambda; \neg\mathsf{student}](X), \mathrm{not}\, DL[\lambda; \mathsf{teacher}](X)$$

with $\lambda = \mathsf{student} \uplus \mathsf{in}_{\mathsf{student}}, \mathsf{student} \uplus \mathsf{in}_{\neg\mathsf{student}}$.

Notice that $DL[\lambda; Q]$ coincides (in this example) with $DL_\Gamma[; Q]$. In fact, [16] already recognizes the need to introduce new predicates in order to write the conclusions of default rules, and these are in turn used to extend the description logic. However, the authors never isolated this construction to study it independently.

There are other important differences. The use of lifting means that the conclusions obtained by means of the default rules are globally reintroduced in the description logic itselft, thus effectively extending the original predicates. In the work mentioned, the authors never explore this possibility: feedback is required in the default rule itself, but nothing is said about the remainder of the program. The programmer has to decide whether and where to add the extension rules to dl-queries – which assumedly has some advantages, but differs from the approach we propose.

Another important difference is the effect of extending a predicate with default rules. Even with the simplifications in [13], one has several distinct predicates to consider: the original predicate P in the description logic, which remains unaffected, and the corresponding predicate in_P about which information will be derived using default rules. The two predicates are only connected at a meta-level (and in the default rule itself), once again leaving the task of merging them to the programmer: even the (intuitive) rule $\mathsf{in}_P \leftarrow DL[; P]$ is missing. Our proposal achieves true tightness, since P and p^+ are simply two different names for the same (semantic) predicate existing at the two levels of the dl-program.

The semantics considered in [16, 13] are answer-set semantics, while we focus on well-founded semantics. We give therefore a different characterization of the semantics of default rules, which does not relate so directly to the Reiter extensions of the logic.

The idea of lifting is addressed briefly in Section 6.1 of [16], in the context of closed-world reasoning. However, yet again the focus is not on the construction itself, but on its application to a specific problem. We feel, therefore, that our work contributes to the field of dl-programs by highlighting this mechanism, studying its properties and showing how it can be used in specific constructions – shedding some insight into similar constructions that had already been arrived at by other authors.

9 Conclusions

This paper shows how the main shortcomings of dl-programs, as discussed in [25], can be overcome by means of a lifting construction that allows predicates to be shared effectively between the two components of a dl-program. In particular, lifting can be used as a first step to implement default rules in dl-programs over predicates defined in the knowledge base. Lifting also brings true tightness to dl-programs, while at the same time making them more flexible.

We demonstrated how integrity constraints can be expressed in dl-programs, either to check their validity or, in the case of full dependencies, to enforce their satisfaction by updating predicates trough lifting.

We also introduced a way to implement default rules in dl-programs and gave characterizations of the semantics of this implementation. We proved an inclusion theorem relating the well-founded semantics

of a dl-program containing default rules with the corresponding Reiter extensions, and provide counter-examples showing that these inclusions are, in general, strict. We intend in the future to strengthen these results, namely by relating well-founded semantics with safe consequences of default rules; and by giving necessary and sufficient conditions for the stated inclusions to become equalities.

The results in these paper have practical consequences, and as such it would be interesting to continue this work in a more experimental setting. We already provided some examples showing that, in practice, our models do give the best expectable results; but an experimental evaluation of these techniques has yet to be undertaken. It is also important to understand how the theoretical, worst-case computational complexity bounds for dl-programs actually affect the actual performance of systems using lifting and default rules, since their strict structure and relative simplicity would suggest their actual behaviour to be better than expected.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] G. Antoniou. A tutorial on default logics. *ACM Computing Surveys*, 31(3):337–359, 1999.

[3] G. Antoniou, C.V. Damasio, 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, 2004. http://rewerse.net/publications/.

[4] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications. 2nd Edition*. Cambridge University Press, 2007.

[5] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14(1):149–180, 1995.

[6] C. Beeri and M.Y. Vardi. The implication problem for data dependencies. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 73–85, London, UK, UK, 1981. Springer-Verlag.

[7] N. Bidoit and C. Froidvaux. Minimalism subsumes default logic and circumscription. In *Proceedings of LICS-87*, pages 89–97, 1987.

[8] A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 14–30, New York, NY, USA, 2009. ACM.

[9] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, pages 260–271, New York, NY, USA, 2003. ACM.

[10] L. Caroprese and M. Truszczyński. Active integrity constraints and revision programming. *Theory Pract. Log. Program.*, 11(6):905–952, November 2011.

[11] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

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

[13] M. Dao-Tran, T. Eiter, and T. Krennwallner. Realizing default logic over Description Logic Knowledge Bases. In C. Sossai and G. Chemello, editors, *10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2009)*, volume 5590 of *Lecture Notes in Artificial Intelligence*, pages 602–613. Springer, July 2009.

[14] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. AL-log: Integrating Datalog and description logics. *Int. Inf. Systems*, 1998.

[15] T. Eiter, G. Ianni, T. Lukasiewicz, and R. Schindlauer. Well-founded semantics for description logic programs in the semantic Web. *ACM Transactions on Computational Logic*, 12(2), 2011. Article 11.

[16] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12–13):1495–1539, 2008.

[17] 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, Lisbon, Portugal, September 4-8, 2006, Tutorial Lectures*, volume 4126 of *Lecture Notes in Computer Science*, pages 93–127. Springer, September 2006.

[18] M. Gelfond. On stratified autoepistemic theories. In *Proceedings of AAAI-87*, pages 207–211, 1987.

[19] S. Heymans, T. Eiter, and G. Xiao. Tractable reasoning with DL-programs over Datalog-rewritable description logics. In H. Coelho, R. Studer, and M. Wooldridge, editors, *Proceedings of 19th European Conference on Artificial Intelligence (ECAI)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 35–40. IOS Press, 2010.

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

[21] V. Lifschitz. Nonmonotonic databases and epistemic queries. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 91)*, pages 381–386, Sydney, Australia, August 1991. Morgan Kaufmann.

[22] F. Lin and Y. Shoham. Epistemic semantics for fixed-points non-monotonic logics. In *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge*, TARK'90, pages 111–120, San Francisco, CA, USA, 1990. Morgan Kaufmann.

[23] B. Motik, I. Horrocks, and U. Sattler. Adding integrity constraints to OWL. In C. Golbreich, A. Kalyanpur, and B. Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[24] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2), 2011.

[25] B. Motik and R. Rosati. Reconciling description logics and rules. *Journal of the ACM*, 57, June 2010.

[26] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, July 2005.

[27] R. Rosati. On the decidability and complexity of integrating ontologies and rules. *Web Semantics Journal*, 2005.

[28] R. Rosati. DL+log: Tight integration of description logics and disjunctive Datalog. In P. Doherty, J. Mylopoulos, and C.A. Welty, editors, *Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 67–78. AAAI Press, June 2006.

[29] J. Tao, E. Sirin, J. Bao, and D.L. McGuinness. Extending OWL with integrity constraints. In V. Haarslev, D. Toman, and G.E. Weddell, editors, *Description Logics*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[30] J. Tao, E. Sirin, J. Bao, and D.L. McGuinness. Integrity constraints in OWL. In M. Fox and D. Poole, editors, *AAAI*. AAAI Press, 2010.

[31] J.D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.