Viewing dl-programs as multi-context systems

Luís Cruz-Filipe,Rita Henriques and Isabel Nunes

DI–FCUL–TR–2013–05

DOI:10455/6895

(http://hdl.handle.net/10455/6895)

April 2013

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.

Viewing dl-programs as multi-context systems

Luís Cruz-Filipe Rita Henriques Isabel Nunes

April 18, 2013

Abstract

The combination of logic programs and description logic knowledge bases has been a fertile topic of research in the last years, with the proposal of several different systems that achieve this goal. In this paper, we look at two of these mechanisms, dl-programs and multi-context systems, which address different aspects of this combination, and include different, incomparable programming constructs. Despite this, we show that every dl-program can be transformed into a multi-context system in such a way that the different semantics for each paradigm are naturally related. As a consequence, many useful constructions developed within the framework of dl-programs may be automatically translated to equivalent constructions in the setting of multi-context systems.

1 Introduction

Several approaches combining rules and ontologies have been proposed in the last years for semantic web reasoning, e.g. [2, 8, 9, 10, 11, 13] among others. Ontologies are typically expressed through decidable fragments of function-free first-order logic with equality, offering a very good ratio expressiveness/complexity of reasoning [1]. The addition of some kind of rule capability in order to be able to express more powerful queries together with nonmonotonic features (in particular, the negation-as-failure operator **not**) achieved by joining ontologies and logic programming result in a very powerful framework for semantic web reasoning.

In this paper, we look at two of these systems: dl-programs [9, 10] and multi-context systems (MCSs) [2], which address different aspects of this combination, and include incomparable programming constructs. One of the main differences is the structure of programs – a dl-program is essentially a logic program that can query a description logic knowledge base Σ and may "feed" its view of Σ with newly inferred facts, while MCSs consist of several knowledge bases, with no restriction on the underlying languages, each declaring additional rules that allow communication with the others. Moreover, dl-programs only allow one knowledge base at a time, expressed in a particular description logic, while MCSs support several knowledge bases, possibly expressed in different languages.

Despite these differences, we show that every dl-program can be transformed in a multi-context system in such a way that different semantics for each paradigm are naturally related: answer-set semantics become grounded equilibria, whereas well-founded semantics correspond to well-founded belief sets. As a consequence, many useful constructions developed within the framework of dl-programs may be automatically translated to equivalent constructions in the setting of MCSs. Although this transformation is intuitive, and has been informally described earlier, the contribution of this work is not only in making it precise, but especially in studying its theoretical properties, namely regarding semantic aspects, and discussing some practical implications. To the authors' knowledge, these aspects have never been addressed before.

The structure of the paper is as follows. Section 2 presents the two systems directly related to our presentation: dl-programs and multi-context systems. In Section 3, we present a translation from the former to the latter and prove several results relating their models. Section 4 proves equivalences between the semantics of both systems, and Section 5 presents and discusses some generalizations and applications of the previous results. Finally Section 6 concludes with some thoughts on the implications of our work.

2 Background

2.1 Description logic programs

In the last years, much effort has been put into combining description logics with rule-based reasoning systems, an approach that is well-suited to modular, independent development of multi-component systems.

One of the proposals in this direction was the introduction of dl-programs [9, 10]. A dl-program is a pair $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$, where Σ is a description logic knowledge base, which we will refer to simply as "knowledge base" from this point onwards, and $\mathcal{P}$ is a generalized logic program – a function-free logic program with negation-as-failure, extended with dl-atoms in rules. Formally, $\mathcal{P}$ is a set of rules of the form $a \leftarrow b_1, \ldots, b_k, \textbf{not}\ b_{k+1}, \ldots, \textbf{not}\ b_m$, called *dl-rules*, where a is an atom and $b_1, \ldots, b_m$ are atoms or dl-atoms. 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 or role of Σ, or a special symbol in $\{=, \neq\}$; $op_i \in \{\uplus, \cup\!\!\!\!-\}$; each p_i is a unary or binary predicate symbol of $\mathcal{P}$ depending on the corresponding S_i being a concept or a role; and $Q(t)$ 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. The sequence $S_1\ op_1\, p_1, \ldots, S_m\ op_m\, p_m$ is the *input context* of the dl-atom. We will use the greek letter χ to denote generic input contexts.

The operators $\uplus$ and $\cup\!\!\!\!-$ are used to extend the knowledge base Σ in the context of the current dl-query. Intuitively, $S_k \uplus p_k$ (resp., $S_k \cup\!\!\!\!-\ p_k$) increases S_k (resp., $\neg S_k$) by the extension of p_k before evaluating the query. (This does not change Σ, only affecting $\mathcal{P}$'s current view of Σ.)

The two components of a dl-program are kept independent, communicating only through dl-atoms. So, although the two components function separately, giving dl-programs nice modularity properties, there is a bidirectional flow of information via dl-atoms. We illustrate the use of dl-programs by means of the following example from [11].

Example 1 *Consider the dl-program $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$, where:*

$$\begin{aligned}
\Sigma: \quad & (\geq 2\mathsf{paperToReview}.\top) \sqsubseteq \mathsf{Overloaded} && (i_1)\\
& \mathsf{Overloaded} \sqsubseteq \forall \mathsf{supervises}^+.\mathsf{Overloaded} && (i_2)\\
& \{(\mathsf{a},\mathsf{b})\} \sqcup \{(\mathsf{b},\mathsf{c})\} \sqsubseteq \mathsf{supervises} && (i_3)
\end{aligned}$$

$$\begin{aligned}
\mathcal{P}: \quad \mathsf{good}(X) \leftarrow & DL[;\mathsf{supervises}](X,Y), \\
& \mathbf{not}\, DL[\mathsf{paperToReview} \uplus \mathsf{paper}; \mathsf{Overloaded}](Y) && (r_1)\\
\mathsf{overloaded}(X) \leftarrow & \mathbf{not}\, \mathsf{good}(X) && (r_2)\\
\mathsf{paper}(\mathsf{b},\mathsf{p_1}) \leftarrow & && (r_3)\\
\mathsf{paper}(\mathsf{b},\mathsf{p_2}) \leftarrow & && (r_4)
\end{aligned}$$

We briefly recall this program's intended meaning as explained in [11]. Axioms (i_1) and (i_2) indicate that someone who has more than two papers to review is overloaded, and that an overloaded person causes all their supervised persons to be overloaded as well. Axiom (i_3) defines the supervision hierarchy. Rule (r_1) indicates that, if X is supervising Y and Y is not overloaded, then X is a good manager. Rule (r_2) indicates that, if X is not a good manager, then X is overloaded.

In order to provide semantics for dl-programs, we first recall the notion of Herbrand base of a(n extended) logic program $\mathcal{P}$, denoted $\mathsf{HB}_{\mathcal{P}}$, the set of all ground atoms consisting of predicate symbols and terms that occur in $\mathcal{P}$. The Herbrand base of a dl-program $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$, denoted $\mathsf{HB}_{\mathcal{KB}}$, is similarly defined, except that constant symbols may also come from the vocabulary of Σ. An interpretation I of $\mathcal{KB}$ relative to $\mathcal{P}$ is any subset of $\mathsf{HB}_{\mathcal{KB}}$; I is a model of, or satisfies, a ground atom or dl-atom a under Σ, denoted $I \models_\Sigma a$, if the following holds:

- if $a \in \mathsf{HB}_{\mathcal{KB}}$ then $I \models_\Sigma a$ iff $a \in I$;
- if a is a ground dl-atom $DL[\chi; Q](t)$ where $\chi = P_1\, op_1\, p_1, \ldots, P_m\, op_m\, p_m$, then $I \models_\Sigma a$ iff $\Sigma(I;\chi) \models Q(t)$, where $\Sigma(I;\chi) = \Sigma \cup \bigcup_m^{i=1} A_i(I)$ and, for $1 \leq i \leq m$,

$$A_i(I) = \begin{cases} \{S_i(e) \mid p_i(e) \in I\}, \text{if } op_i = \uplus \\ \{\neg S_i(e) \mid p_i(e) \in I\}, \text{if } op_i = \mathbin{\cup\!\!\!\!-} \end{cases}$$

An interpretation I is a *model* of a ground dl-rule r iff $I \models_\Sigma H(r)$ whenever $I \models_\Sigma B(r)$, where $H(r)$ and $B(r)$ are the head and the body of rule r, respectively; I is a model of $\mathcal{KB}$ iff I is a model of all ground rules of $\mathcal{P}$.

Example 2 *Given the dl-program $\mathcal{KB}$ of Example 1, its Herbrand base will be*

$$\mathsf{HB}_{\mathcal{KB}} = \{\mathsf{good}(t), \mathsf{overloaded}(t) \mid t \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{p}_1, \mathsf{p}_2\}\}\{\mathsf{paper}(t_1, t_2) \mid t_1, t_2 \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{p}_1, \mathsf{p}_2\}\} .$$

This may seem a bit strange, since e.g. $\mathsf{paper}(\mathsf{a}, \mathsf{c})$ *or* $\mathsf{overloaded}(\mathsf{p}_1)$ *do not fit well with our intended interpretation of the predicates* paper *and* $\mathsf{overloaded}$*; but this is a side-effect of the absence of types in logic programming.*

Examples of interpretations for $\mathcal{KB}$ are:

- *the empty set $I_1 = \emptyset$;*
- *the set $I_2 = \{\mathsf{paper}(\mathsf{b}, \mathsf{p}_1), \mathsf{paper}(\mathsf{b}, \mathsf{p}_2), \mathsf{overloaded}(\mathsf{a})\}$;*
- *or $I_3 = \{\mathsf{paper}(\mathsf{a}, \mathsf{c}), \mathsf{overloaded}(\mathsf{b}), \mathsf{good}(\mathsf{p}_2)\}$;*
- *or even $I_4 = \{\mathsf{paper}(\mathsf{b}, \mathsf{p}_1), \mathsf{paper}(\mathsf{b}, \mathsf{p}_2)\} \cup \{\mathsf{overloaded}(t) \mid t \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{p}_1, \mathsf{p}_2\}\}$.*

It is easy to verify that only I_4 is a model of $\mathcal{KB}$. In particular, I_1 and I_3 do not satisfy rules (r_3) and (r_4), while I_2 does not satisfy rule (r_2) with $X = \mathsf{b}$.

As usual, a dl-program $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$ is *positive* if the rules in $\mathcal{P}$ do not contain negations. Positive dl-programs enjoy the usual properties of positive logic programs, namely they have a unique least model $\mathsf{M}_{\mathcal{KB}}$ that can be constructed by computing the least fixed-point of the Herbrand transformation $\mathsf{T}_{\mathcal{KB}}$, which is defined as the usual Herbrand transformation for logic programs, resorting to Σ to evaluate dl-atoms. The dl-program in Example 1 is not a positive program because of rules (r_1) and (r_2).

Answer-set semantics. The answer set semantics of (not necessarily positive) dl-programs is defined again in analogy to that of logic programs. There are two possible generalizations, yielding *strong* and *weak* answer set semantics; in this paper, we will only use the former, and usually omit the adjective "strong".

Given a dl-program $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$, we can obtain a positive dl-program by replacing $\mathcal{P}$ with its *strong dl-transform* $s\mathcal{P}^I_\Sigma$ relative to Σ and an interpretation $I \subset \mathsf{HB}_{\mathcal{P}}$. This is obtained by grounding every rule in $\mathcal{P}$ and then (i) deleting every dl-rule r such that $I \models_\Sigma a$ for some default negated a in the body of r, and (ii) deleting from each remaining dl-rule the negative body. The informed reader will recognize this to be a generalization of the Gelfond–Lifschitz reduct. Since $\mathcal{KB}^I = \langle \Sigma, s\mathcal{P}^I_\Sigma \rangle$ is a positive dl-program, it has a unique least model $\mathsf{M}_{\mathcal{KB}^I}$. A *strong answer set* of $\mathcal{KB}$ is an interpretation I that coincides with $\mathsf{M}_{\mathcal{KB}^I}$.

Example 3 *Consider again the dl-program $\mathcal{KB}$ and the interpretations I_1, I_2, I_3 and I_4 defined above. We can verify that I_4 is an answer set for $\mathcal{KB}$.*

First, we need to compute $s\mathcal{P}^{I_4}_\Sigma$. Consider the ground instances of (r_1); there are only two cases when $I_4 \models DL[;\mathsf{supervises}](X, Y)$: when X is a *and Y is* b*, or when X is* b *and Y is* c*, so all other ground instances of (r_1) will be removed from $s\mathcal{P}^{I_4}_\Sigma$. Furthermore, $I_4 \models DL[\mathsf{paperToReview} \uplus \mathsf{paper}; \mathsf{Overloaded}](\mathsf{b})$ due to axiom (i_1) from Σ, and $I_4 \models$*

$DL[\mathsf{paperToReview} \uplus \mathsf{paper}; \mathsf{Overloaded}](\mathsf{c})$ by (i_1) and (i_2). Therefore, $s\mathcal{P}_\Sigma^{I_4}$ contains no ground instances of (r_1).

As regards (r_2), since I_4 does not contain $\mathsf{good}(t)$ for any t, all its ground instances will be included in $s\mathcal{P}_\Sigma^{I_4}$ with their body removed. Finally, (r_3) and (r_4) are grounded rules with no negative literals in their bodies, so they are copied to $s\mathcal{P}_\Sigma^{I_4}$ unchanged. Thus, $s\mathcal{P}_\Sigma^{I_4}$ is the following program.

$$\begin{array}{cccc}\mathsf{overloaded}(\mathsf{a}) & \mathsf{overloaded}(\mathsf{b}) & \mathsf{overloaded}(\mathsf{c}) & \mathsf{overloaded}(\mathsf{p}_1) \\ \mathsf{overloaded}(\mathsf{p}_2) & \mathsf{paper}(\mathsf{b},\mathsf{p}_1) & \mathsf{paper}(\mathsf{b},\mathsf{p}_2) & \end{array}$$

Since $s\mathcal{P}_\Sigma^{I_4}$ contains only facts, it is easy to compute the least model of $\mathcal{KB}^{I_4}$ and check that it coincides with I_4. Therefore, I_4 is an answer set for $\mathcal{KB}$.

By comparison, consider the interpretation I_2. It is easy to verify that $s\mathcal{P}_\Sigma^{I_2}$ coincides with $s\mathcal{P}_\Sigma^{I_4}$ computed above, since in this case the construction of the reduct only depends on the instances of paper *and* good *included in the interpretation. Therefore, the least model of $\mathcal{KB}^{I_2}$ is again I_4, and thus I_2 is* not *an answer set for $\mathcal{KB}$. The reader is invited to verify that I_4 is actually the only answer set for this dl-program.*

An algorithm for computing strong (and weak) answer sets of dl-programs has been implemented in the DL-plugin for `dlvhex` [7].

WELL-FOUNDED SEMANTICS. Another possible semantics for dl-programs is well-founded semantics, which again generalizes well-founded semantics for logic programs. There are several equivalent ways to define this semantics; for the purpose of this paper, we define the well-founded semantics of a dl-program $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$ by means of the operator $\gamma_{\mathcal{KB}}$ such that $\gamma_{\mathcal{KB}}(I)$ is the least model of the positive dl-program $\mathcal{KB}^I$ defined above.

This operator is anti-monotonic (if $I \subseteq J$, then $\gamma_{\mathcal{KB}}(I) \supseteq \gamma_{\mathcal{KB}}(J)$), so $\gamma^2_{\mathcal{KB}}$ is monotonic and therefore it has a least and greatest fixpoint, denoted $\mathsf{lfp}\,(\gamma^2_{\mathcal{KB}})$ and $\mathsf{gfp}\,(\gamma^2_{\mathcal{KB}})$, respectively. An atom $a \in \mathsf{HB}_\mathcal{P}$ is *well-founded* if $a \in \mathsf{lfp}\,(\gamma^2_{\mathcal{KB}})$ and *unfounded* if $a \notin \mathsf{gfp}\,(\gamma^2_{\mathcal{KB}})$; the *well-founded semantics* of $\mathcal{KB}$ is the set containing all well-founded atoms and the negations of all unfounded atoms. Intuitively, well-founded atoms are true in every model of $\mathcal{P}$, whereas unfounded atoms are always false. Note that, unlike answer sets, the well-founded semantics of $\mathcal{KB}$ may not be a model of $\mathcal{KB}$.

Example 4 *We quickly summarize the essential steps in computing the well-founded semantics of the dl-program $\mathcal{KB}$ from Example 1.*

First, we compute the least fixed point of $\gamma^2_{\mathcal{KB}}$. Beginning with $J_0 = \emptyset$, we compute $s\mathcal{P}_\Sigma^{J_0}$. This program contains two closed instances of rule (r_1) for which $J_0 \models DL[;\mathsf{supervises}](X,Y)$, namely those with $X = \mathsf{a}$ and $Y = \mathsf{b}$ or $X = \mathsf{b}$ and $Y = \mathsf{c}$.[1] Also, $s\mathcal{P}_\Sigma^{J_0}$ contains the facts overloaded(t) *for all t (since J_0 does not contain* good(t) *for any t) and the rules (r_3) and (r_4). The least model of $\mathcal{KB}^{J_0}$ is*

$$J_1 = \{\mathsf{good}(\mathsf{a}), \mathsf{good}(\mathsf{b}), \mathsf{paper}(\mathsf{b},\mathsf{p}_1), \mathsf{paper}(\mathsf{b},\mathsf{p}_2)\}\{\mathsf{overloaded}(t) \mid t \in \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{p}_1,\mathsf{p}_2\}\}\,.$$

[1]Note that $J_0 \not\models DL[\mathsf{paperToReview} \uplus \mathsf{paper}; \mathsf{Overloaded}](Y)$ for any Y, since J_0 contains no facts about paper.

Now we compute $s\mathcal{P}_\Sigma^{J_1}$. This contains no instances of (r_1): the two instances that were included before are now removed because $J_1 \models DL[\mathsf{paperToReview} \uplus \mathsf{paper}; \mathsf{Overloadedloaded}](Y)$ for $Y = \mathsf{b}$ and $Y = \mathsf{c}$. Furthermore, since $J_1 \models \mathsf{good}(a)$ and $J_1 \models \mathsf{good}(b)$, $s\mathcal{P}_\Sigma^{J_1}$ contains only the facts $\mathsf{overloaded}(\mathsf{c})$, $\mathsf{overloaded}(\mathsf{p_1})$ and $\mathsf{overloaded}(\mathsf{p_2})$ as instances of (r_2). The two last rules are unchanged. The least model of $\mathcal{KB}^{J_1}$ is now

$$J_2 = \{\mathsf{paper}(\mathsf{b}, \mathsf{p_1}), \mathsf{paper}(\mathsf{b}, \mathsf{p_2}), \mathsf{overloaded}(\mathsf{c}), \mathsf{overloaded}(\mathsf{p_1}), \mathsf{overloaded}(\mathsf{p_2})\}$$

and this coincides with $\gamma^2_{\mathcal{KB}}(J_0)$.

Continuing this process, we compute $s\mathcal{P}_\Sigma^{J_2}$, which contains no instances of (r_1) as in the previous case, but includes all closed instances of $\mathsf{overloaded}(t)$ since J_2 again contains no facts about good. The least model of $\mathcal{KB}^{J_2}$ is

$$J_3 = \{\mathsf{paper}(\mathsf{b}, \mathsf{p_1}), \mathsf{paper}(\mathsf{b}, \mathsf{p_2})\} \cup \{\mathsf{overloaded}(t) \mid t \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{p_1}, \mathsf{p_2}\}\} .$$

Finally, $s\mathcal{P}_\Sigma^{J_3}$ can be seen to coincide with $s\mathcal{P}_\Sigma^{J_2}$, and therefore $J_3 = \mathsf{lfp}\,(\gamma_{\mathcal{KB}})$; hence, $J_3 = \gamma_{\mathcal{KB}}(J_3) = \gamma^2_{\mathcal{KB}}(J_2)$ is also the least fixed point of $\gamma^2_{\mathcal{KB}}$.

The greatest fixed point of $\gamma^2_{\mathcal{KB}}$ is actually easier to compute, and coincides again with J_3. Therefore, the well-founded semantics of $\mathcal{KB}$ contains all atoms in J_3 together with the negations of all other closed atoms in $\mathsf{HB}_{\mathcal{KB}}$. In this case, all closed atoms are either well-founded or unfounded, which in particular means that there is only one answer set (the set I_4 mentioned earlier, which coincides with J_3) and the positive part of this well-founded semantics (namely, J_3) is a model of $\mathcal{KB}$ – see Theorem 5.9 of [9] for details.

2.2 Multi-context systems

More recently, *multi-context systems* [2] have been proposed as another, more general approach at combining different reasoning paradigms. A multi-context system allows information among different logics to flow within the system through bridge rules.

Within this setting, a *logic* is defined as a triple $\Sigma = (KB_\Sigma, BS_\Sigma, ACC_\Sigma)$ where KB_Σ is the set of well-formed knowledge bases of Σ, BS_Σ is the set of possible belief sets, and $ACC_\Sigma : KB_\Sigma \to 2^{BS_\Sigma}$ is a function describing the semantics of the logic by assigning to each element of KB_Σ a set of acceptable sets of beliefs. Note that nothing is said about *what* knowledge bases or belief sets are; the former are part of the syntax of the language, their precise definition being left to Σ, while the latter intuitively represent the sets of syntactical elements representing the beliefs an agent may adopt. Still, this definition is meant to be abstract and general, so part of the purpose of KB_Σ and BS_Σ is defining these notions for each logic Σ..

Given a set of logics $\Sigma = \{\Sigma_1, \ldots, \Sigma_n\}$, a Σ_k-*bridge rule*, with $1 \leq k \leq n$, has the form $s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \mathbf{not}(r_{j+1} : p_{j+1}), \ldots, \mathbf{not}(r_m : p_m)$, where $1 \leq r_k \leq n$, p_k is an element of some belief set of Σ_{r_k}, and $kb \cup \{s\} \in KB_k$ for each $kb \in KB_k$. A multi-context system (MCS) $M = \langle C_1, \ldots, C_n \rangle$ is a collection of contexts $C_i = (\Sigma_i, kb_i, br_i)$ where $\Sigma_i = (KB_i, BS_i, ACC_i)$ is a logic, kb_i is a knowledge base (an element of KB_i) and br_i is a set of Σ_i-bridge rules over $\{\Sigma_1, \ldots, \Sigma_n\}$.

Since bridge rules refer to other contexts, they may add information to a context based on beliefs in other contexts. Considering fixed the above MCS $M = \langle C_1, \dots, C_n \rangle$, a bridge rule is *applicable* in a *belief state* $S = \langle S_1, \dots, S_n \rangle$, where each S_i is an element of BS_i, iff $p_i \in S_{r_i}$ for $1 \leq i \leq j$ and $p_k \notin S_{r_k}$ for $j+1 \leq k \leq m$. The semantics of MCSs is based upon the concept of *equilibrium*, which intuitively represents a belief state of the MCS where for each context C_i the selected belief set is among the acceptable belief sets for C_i's knowledge base together with the heads of C_i's applicable bridge rules. Formally, a belief state $S = \langle S_1, \dots, S_n \rangle$ of M is an equilibrium iff the condition $S_i \in ACC_i(kb_i \cup \{head(r) \mid r \in br_i \text{ applicable in } S\})$ holds for $1 \leq i \leq n$. An equilibrium is *minimal* if it is not a superset of any other equilibrium.

Here again, the reduct of the program underlying an MCS, which is a positive program, is used to compute the MCS's minimal model. Reducts are calculated over *reducible* MCSs, so first we define the notion of reducibility w.r.t. a logic, a context and, finally, an MCS.

A logic $\Sigma = (KB_\Sigma, BS_\Sigma, ACC_\Sigma)$ is called reducible if (1) there is $KB^*_\Sigma \subseteq KB_\Sigma$ such that the restriction of Σ to KB^*_Σ is monotonic[2]; (2) there is a reduction function $red_\Sigma : KB_\Sigma \times BS_\Sigma \rightarrow KB^*_\Sigma$ such that for each $k \in KB_\Sigma$ and $S, S' \in BS_\Sigma$: (a) $red_\Sigma(k, S) = k$ whenever $k \in KB^*_\Sigma$; (b) red_Σ is anti-monotonic in the second argument; and (c) $S \in ACC_\Sigma(k)$ iff $ACC_\Sigma(red_\Sigma(k, S)) = \{S\}$.

A context $C = (\Sigma, kb, br)$ is reducible if (1) its logic is reducible; (2) for all $H \subseteq \{head(r) \mid r \in br\}$ and belief sets S, $red_\Sigma(kb \cup H, S) = red_\Sigma(kb, S) \cup H$. A multi-context system is reducible if all of its contexts are reducible.

Grounded equilibria. A *definite* MCS is a reducible MCS in which bridge rules are monotonic (that is, they do not contain **not**) and knowledge bases are in reduced form (that is, $kb_i = red_i(kb_i, S)$ for all i and every $S \in BS_i$). Every definite MCS has a unique minimal equilibrium [2], which we will denote by $Eq(M)$.

For non-definite MCSs, we resort to a generalization of the Gelfond–Lifschitz reduct to the multi-context case. If $M = \langle C_1, \dots, C_n \rangle$ is a reducible MCS and $S = \langle S_1, \dots, S_n \rangle$ is a belief state of M, the *S-reduct* of M is $M^S = \langle C^S_1, \dots, C^S_n \rangle$, such that $C^S_i = (\Sigma_i, red(kb_i, S_i), br^S_i)$. Here, br^S_i results from br_i by deleting (1) every rule with some **not** $(k : p)$ in the body such that $p \in S_k$, and (2) all **not** literals from the bodies of remaining rules. In [2], it is proved that M^S is definite; if $S = Eq(M^S)$, then S is a *grounded equilibrium* of M.

If M is a definite MCS, then its minimal equilibrium is its only grounded equilibrium; in other cases, several grounded equilibria (or none) may exist. It is also easy to verify that grounded equilibria of M are indeed equilibria of M.

Well-founded semantics. The well-founded semantics for reducible MCSs is also defined in [2], and is based on the operator $\gamma_M(S) = Eq(M^S)$, defined for each M such that, for each logic Σ_i in any of M's contexts, BS_i has a least element. Here again, γ_M is anti-monotonic, so γ^2_M is monotonic and therefore it has a least fixpoint, denoted $\mathsf{lfp}\,(\gamma^2_M)$. Following what is done for logic programs, the well-founded semantics of M – $WFS(M)$ – is $\mathsf{lfp}\,(\gamma^2_M)$. It should be pointed out that WFS(M) is not necessarily an equilibrium: informally, it contains the

[2]A logic Σ is said to be monotonic if $ACC_\Sigma(kb)$ is always a singleton set, and $kb \subseteq kb'$ implies that the only element of $ACC_\Sigma(kb)$ is a subset of the only element of $ACC_\Sigma(kb')$. This coincides with the usual notion of monotonic logic.

knowledge that is common to all equilibria, but being an equilibrium is not preserved by intersection. One can easily obtain an example by picking a dl-program whose well-founded semantics is not an equilibrium (see e.g. [4]) and applying the translation we define in the next section.

The next section will detail how dl-programs can be translated in MCSs, illustrating this construction with the MCS generated from the dl-program in Example 1 above and providing examples of the different concepts introduced at this stage.

Several other mechanisms for combining rules and ontologies have been proposed, but are outside the scope of this paper. The interested reader can find a more comprehensive overview of the different approaches in the introduction of [12].

3 From dl-programs to multi-context systems

There are two essential differences between dl-programs and multi-context systems: the former only allow the combination of a logic program with a description logic knowledge base, whereas the latter allow any number of different systems to be joined together; but the former allow for *local* changes to the knowledge base, via input contexts[3] in dl-atoms, whereas the interaction within the latter is *global*, since bridge rules do add new inferences to the context.

Because of these two differences, the two systems have different kinds of expressiveness. In spite of this, we can generate a multi-context system $M = \mathsf{M}(\mathcal{KB})$ from a given dl-program $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$. There are two steps in this process.

1. We split $\mathcal{P}$ in its purely logical part and its communication part, translating rules that contain dl-atoms into bridge rules.
2. For each distinct input context χ appearing in $\mathcal{P}$, we create a different copy of the knowledge base, corresponding to the view of the knowledge base within the dl-atoms containing χ.

Although the essential idea behind this construction is already suggested in [3], the authors' purpose in that paper is simply to justify that multi-context systems are a generalization of dl-programs, and therefore they do not study the construction in detail. Our goal is different, however, so we begin by making this definition precise.

Definition 1 *Let $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$ be a dl-program and $\chi_1, \ldots, \chi_n$ be the distinct input contexts occurring in dl-atoms in $\mathcal{P}$.*

1. *The translation $\sigma_{\mathcal{KB}}$ of literals and dl-atoms is defined as follows: if L is a literal, then $\sigma_{\mathcal{KB}}(L) = (0 : L)$; for dl-atoms, $\sigma_{\mathcal{KB}}(DL[\chi_i; Q](t)) = (i : Q(t))$; and for their negations $\sigma_{\mathcal{KB}}(\mathbf{not}\, DL[\chi_i; Q](t)) = \mathbf{not}(i : Q(t))$.*

   The translation of $\mathcal{P}$ is the context $C_0 = \langle L_0, kb_0, br_0 \rangle$ where:

[3]Recall that, if $DL[\chi; Q](t)$ is a dl-atom, χ is its input context and Q is its dl-query.

- $L_0 = \langle KB_0, BS_0, ACC_0\rangle$ *is the logic underlying* $\mathcal{P}$*, where* KB_0 *is the set of all logic programs over* $\mathcal{P}$*'s signature,* BS_0 *is the power set of* $\mathsf{HB}_{\mathcal{P}}$*, and* ACC_0 *assigns each program to the set of its models;*
- kb_0 *is* $\mathcal{P}^-$*, the set of rules of* $\mathcal{P}$ *that do not contain any dl-atoms;*
- br_0 *contains* $p \leftarrow \sigma_{\mathcal{KB}}(l_1), \ldots, \sigma_{\mathcal{KB}}(l_m)$ *for each rule* $p \leftarrow l_1, \ldots, l_m$ *in* $\mathcal{P} \setminus \mathcal{P}^-$*.*

2. *For each input context* $\chi_i = P_1\, op_1\, p_1, \ldots, P_k\, op_k\, p_k$*, with* $i = 1, \ldots, n$*, the context* $C_i = \langle L, kb, br_i\rangle$ *is defined as follows.*

   - $L = \langle KB, BS, ACC\rangle$ *is the description logic underlying* Σ*, with* KB *the set of all ontologies over* Σ*'s signature;* BS *contains all sets of dl-queries to* Σ*; and* ACC *assigns to each ontology the set of dl-queries it satisfies.*[4]
   - kb *is* Σ*.*
   - *For* $j = 1, \ldots, k$*,* br_i *contains* $P_j \leftarrow (0 : p_j)$*, if* $op_j = \uplus$*, or* $\neg P_j \leftarrow (0 : p_j)$*, if* $op_j = \cup\!\!\!\!-$*.*

   Note that L *and* kb *are the same for all contexts originating from* Σ*.*

3. *The multi-context system* generated *by* $\mathcal{KB}$ *is* $\mathsf{M}(\mathcal{KB}) = \langle C_0, C_1, \ldots, C_n\rangle$*.*

The first context in $\mathsf{M}(\mathcal{KB})$ is a logic program with the same underlying language of $\mathcal{P}$. This implies that any interpretation I of $\mathcal{P}$ is an element of BS_0, and vice-versa. We will use this fact hereafter without mention.

Example 5 *Recall the dl-program* $\mathcal{KB}$ *from Example 1. For the purpose of generating an MCS from* $\mathcal{KB}$*, observe that there are two different input contexts in this program,* $\chi_1 = \epsilon$ *and* $\chi_2 = \mathsf{paperToReview} \uplus \mathsf{paper}$*. Also,* (r_1) *is the only rule in* $\mathcal{P}$ *containing dl-atoms, so it will be the only rule not present in* $C_0 = \mathcal{P}^-$*.*

The generated multi-context system $\mathsf{M}(\mathcal{KB})$ *is* $\langle C_0, C_1, C_2\rangle$*, where:*

- $C_0 = \langle L_0, \{r_2, r_3, r_4\}, \{\mathsf{good}(X) \leftarrow (1 : \mathsf{supervises}(X, Y)), \mathbf{not}(2 : \mathsf{Overloaded}(Y))\}\rangle$
- $C_1 = \langle L, \Sigma, \emptyset\rangle$
- $C_2 = \langle L, \Sigma, \{\mathsf{paperToReview}(X, Y) \leftarrow (0 : \mathsf{paper}(X, Y))\}\rangle$

Just as we can generate a multi-context system $\mathsf{M}(\mathcal{KB})$ from any dl-program $\mathcal{KB}$, we can generate a belief state for $\mathsf{M}(\mathcal{KB})$ from any interpretation of $\mathcal{KB}$.

Definition 2 *Let* $\mathcal{KB} = \langle \Sigma, \mathcal{P}\rangle$ *be a dl-program and* I *an interpretation of* $\mathcal{KB}$*. The belief state* generated *by* I *is* $\mathsf{S}_{\mathcal{KB}}(I) = \langle S_0^I, S_1^I, \ldots, S_n^I\rangle$ *of* $\mathsf{M}(\mathcal{KB})$*, where* $S_0^I = I$ *and, for* $i = 1, \ldots, n$*,*

$$S_i^I = ACC(\Sigma \cup \{P(t) \mid I \models p(t), P \uplus p \in \chi_i\} \cup \{\neg P(t) \mid I \models p(t), P \cup\!\!\!\!- \; p \in \chi_i\}) \,.$$

[4]Formally, we can define ACC as computing the set of logical consequences of the ontology and restricting it to those formulas that are dl-queries.

It is straightforward to verify that $\mathsf{S}_{\mathcal{KB}}(I)$ is a belief state of $\mathsf{M}(\mathcal{KB})$. When there is only one dl-program under consideration, we omit the subscript in $\mathsf{S}_{\mathcal{KB}}$.

Example 6 *Recall the interpretations for the dl-program $\mathcal{KB}$ from Example 2. The belief states generated by these interpretations all contain the interpretation itself as belief set for C_0 and $B = \{\mathsf{supervises(a, b)}, \mathsf{supervises(b, c)}\}$ as belief set for C_1 (since $\chi_1 = \emptyset$ and this is the knowledge base corresponding to Σ). In the case of I_1, this is also the belief set for C_2, since I_1 satisfies no input predicates in χ_2. Thus, $\mathsf{S}(I_1) = \langle \emptyset, B, \{\mathsf{supervises(a, b)}, \mathsf{supervises(b, c)}\}\rangle$.*

Interpretation I_2 satisfies $\mathsf{paper(b, t_1)}$ and $\mathsf{paper(b, t_1)}$, whence $\mathsf{paperToReview(b, t_1)}$ and $\mathsf{paperToReview(b, t_2)}$ will be included in the belief set for C_2, together with its consequences – namely, $\mathsf{Overloaded(b)}$ and $\mathsf{Overloaded(c)}$, applying axioms (i_1) and (i_2), respectively. Thus:

$$\begin{aligned}\mathsf{S}(I_2) = \langle I_2, B, \{&\mathsf{supervises(a, b)}, \mathsf{supervises(b, c)}, \mathsf{paperToReview(b, t_1)},\\ &\mathsf{paperToReview(b, t_2)}, \mathsf{Overloaded(b)}, \mathsf{Overloaded(c)}\}\rangle\end{aligned}$$

The cases of I_3 and I_4 are very similar, and the reader can check that:

$$\begin{aligned}\mathsf{S}(I_3) &= \langle I_3, B, \{\mathsf{supervises(a, b)}, \mathsf{supervises(b, c)}, \mathsf{paperToReview(a, c)}\}\rangle\\ \mathsf{S}(I_4) &= \langle I_4, B, \{\mathsf{supervises(a, b)}, \mathsf{supervises(b, c)}, \mathsf{paperToReview(b, t_1)},\\ &\qquad \mathsf{paperToReview(b, t_2)}, \mathsf{Overloaded(b)}, \mathsf{Overloaded(c)}\}\rangle\end{aligned}$$

In the previous example, one can see that $\mathsf{S}(I_4)$ is the only belief state that is also an equilibrium of $\mathsf{M}(\mathcal{KB})$; recall that I_4 was the only interpretation that was also a model of $\mathcal{KB}$. This suggests that there are very close connections between I and $\mathsf{S}(I)$, which we will now prove formally.

Lemma 1 *Let $\mathcal{KB} = \langle \Sigma, \mathcal{P}\rangle$ be a dl-program and $DL[\chi_i : Q](t)$ be a ground dl-atom in $\mathcal{P}$.*

1. *For any interpretation I, $I \models DL[\chi_i : Q](t)$ iff $Q(t) \in S_i^I$.*
2. *If $S = \langle S_0, S_1, \ldots, S_n\rangle$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$ and $1 \leq i \leq n$, then $Q(t) \in S_i$ iff $S_0 \models DL[\chi_i : Q](t)$.*

Proof. The first equivalence is straightforward, since the construction of S_i^I mimicks the definition of the semantics of $\mathcal{KB}$. The second equivalence is a matter of checking that semantics of dl-programs and the definition of equilibrium are compatible. □

Theorem 1 *Let $\mathcal{KB} = \langle \Sigma, \mathcal{P}\rangle$ be a dl-program.*

1. *If I is a model of $\mathcal{KB}$, then $\mathsf{S}(I)$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$.*
2. *If $S = \langle S_0, \ldots, S_n\rangle$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$, then S_0 is a model of $\mathcal{KB}$.*

Proof.

1. Suppose that I is a model of $\mathcal{KB}$ and let $\mathsf{S}(I) = \langle S_0^I, S_1^I, \ldots, S_n^I \rangle$ be the belief state generated by I. For each i, we need to show that S_i^I is a belief set of kb_i and that $s \in S_i^I$ whenever the bridge rule $s \leftarrow \sigma(l_1), \ldots, \sigma(l_k) \in br_i$ is applicable in $\mathsf{M}(\mathcal{KB})$.

   Consider first the case $i = 0$. Since $S_0^I = I$ is a model of all the rules in $\mathcal{P}$, it follows that I satisfies every rule in $kb_0 = \mathcal{P}^-$. Let $s \leftarrow \sigma_{\mathcal{KB}}(l_1), \ldots, \sigma_{\mathcal{KB}}(l_k)$ be a bridge rule in br_0; this must originate from a rule $s \leftarrow l_1, \ldots, l_k$ in $\mathcal{P} \setminus \mathcal{P}^-$. Assume that the bridge rule is applicable in $\mathsf{S}(I)$; for each l_j, there are three possibilities: (1) l_j is a regular literal, and then $I \models l_j$; (2) l_j is $DL[\chi_m; Q](t)$ and $Q(t) \in S_m^I$, whence $I = S_0^I \models l_j$ by Lemma 1; (3) l_j is **not** $DL[\chi_m; Q](t)$ and $Q(t) \notin S_m^I$, whence $I = S_0^I \not\models DL[\chi_m; Q](t)$ by Lemma 1, and therefore $I \models l_j$. Thus, I satisfies the body of the rule, hence $S_0^I = I \models s$.

   Suppose now that $i \neq 0$. By construction, $S_i^I = ACC(\Sigma \cup \{P(t) \mid I \models p(t), P \uplus p \in \chi_i\} \cup \{\neg P(t) \mid I \models p(t), P \cup\!\!\!\!- \; p \in \chi_i\})$, which is precisely the (unique) model of Σ together with the heads of the bridge rules applicable in $\mathsf{S}(I)$.

   Therefore $\mathsf{S}(I)$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$.

2. Suppose that $S = \langle S_0, S_1, \ldots, S_n \rangle$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$. Since S_0 is a model of kb_0 extended with the heads of bridge rules in br_0 which are applicable in S, it follows that S_0 satisfies all the rules of $kb_0 = \mathcal{P}^-$.

   Let $p \leftarrow l_1, \ldots, l_k$ be a rule in $\mathcal{P} \setminus \mathcal{P}^-$. Then $p \leftarrow \sigma_{\mathcal{KB}}(l_1), \ldots, \sigma_{\mathcal{KB}}(l_k)$ is a bridge rule in br_0. Again, if S_0 satisfies the body of the rule, then the corresponding bridge rule is applicable in S: for regular literals this is immediate (the condition is the same), while for dl-atoms and their negations this is again Lemma 1. Hence S_0 is also a model of the remaining rules in $\mathcal{P}$.

   Therefore S_0 is a model of $\mathcal{KB}$. $\square$

Corollary 1 *If $S = \langle S_0, \ldots, S_n \rangle$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$, then $\mathsf{S}(S_0) = S$.*

This result allows us to state all future equivalences in terms of models of $\mathcal{P}$, since any equilibrium S of $\mathsf{M}(\mathcal{KB})$ is uniquely defined from its first component.

Furthermore, this correspondence preserves inclusions, so we also have the following relationship.

Theorem 2 *Let $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$ be a positive dl-program. Then I is the least model of $\mathcal{KB}$ iff $\mathsf{S}(I)$ is a minimal equilibrium of $\mathsf{M}(\mathcal{KB})$.*

Proof.

($\Rightarrow$) Suppose that I is a least model of $\mathcal{KB}$, and let $S' = \langle S'_0, S'_1, \ldots, S'_n \rangle$ be an equilibrium of $\mathsf{M}(\mathcal{KB})$. By Theorem 1, S'_0 is a model of $\mathcal{P}$, and since I is minimal, it follows that $I \subseteq S'_0$. Suppose also that $\mathsf{S}(I)$ is not minimal; then $S'_0 \subseteq I$; hence $S'_0 = I$. By the corollary proved above, it follows that $S' = \mathsf{S}(I)$, which is a contradiction. Hence $\mathsf{S}(I)$ is minimal.

($\Leftarrow$) Suppose that $\mathsf{S}(I) = \langle S_0^I, S_1^I, \ldots, S_n^I \rangle$ is a minimal equilibrium of $\mathsf{M}(\mathcal{KB})$, and let I' be a model of $\mathcal{KB}$. Let $\mathsf{S}(I')$ be the corresponding equilibrium of $\mathsf{M}(\mathcal{KB})$; then $I = S_0^I \subseteq S_0^{I'} = I'$ by minimality of S, hence I is a least model of $\mathcal{P}$. $\square$

4 Equivalence between semantics

We now show that the semantics for dl-programs have a precise correspondence with the semantics of multi-context systems: strong answer sets are equivalent to grounded equilibria, and the well-founded semantics for both systems are strongly related. This should not come as a big surprise: both the strong-dl transform of dl-programs and the reduct of a multi-context system are generalizations of the Gelfond–Lifschitz transform of ordinary logic programs.

Throughout this section, let $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$ be a dl-program and $\mathsf{M}(\mathcal{KB})$ be the multi-context system generated by $\mathcal{KB}$, where $\mathsf{M}(\mathcal{KB}) = \langle C_0, C_1, \ldots, C_n \rangle$ and $C_i = \langle L_i, kb_i, br_i \rangle$ for $i = 0, \ldots, n$.

Proposition 1 $\mathsf{M}(\mathcal{KB})$ *is reducible, with* KB_0^* *the set of positive programs,* $\mathsf{red}_0(k, S) = k^S$*, computing the Gelfond–Lifschitz transform of* k *relative to* S*,* $KB_i^* = KB_i$ *and* $\mathsf{red}_i(k, S) = k$ *a projection function.*

Proof.

1. The logic L_0 is reducible, with KB_0^* the set of positive programs and reduction function $\mathsf{red}_0(k, S) = k^S$, computing the Gelfond–Lifschitz transform.

   We need to show that the conditions for being a reducible logic hold. These are just well-known facts in logic programming, namely:

   - positive logic programs are monotonic;
   - the Gelfond–Lifschitz transform of a positive logic program is itself;
   - if $S \subseteq S'$, then $k^{S'} \subseteq k^S$;
   - S is a model of a (general) logic program k iff it is a model of k^S.

2. The context C_0 is reducible.

   We need to show that $\mathsf{red}_0(kb \cup H, I) = \mathsf{red}_0(kb, I) \cup H$, for every interpretation I and any set H of heads of rules in br_0. But H consists solely of facts (rules with empty body), which are unaltered by the Gelfond–Lifschitz transform, hence this equality holds.

3. For $i = 1, \ldots, n$, the context C_i is reducible via the identity function.

   Since description logics are monotonic, we can take $KB_i^* = KB_i$, and the identity function trivially satisfies all the reducibility conditions. $\square$

We now look closely at the relationship between the strong dl-transform of $\mathcal{P}$ and the reduction of $\mathsf{M}(\mathcal{KB})$. The former generates a subsystem of the latter in the following sense.

Definition 3 *Consider two multi-context systems $M = \langle C_1, \ldots, C_n \rangle$ and $M' = \langle C'_1, \ldots, C'_m \rangle$. We say that M' is a* subsystem *of M, $M' \subseteq M$, if there exists an injective function $\varphi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that C'_i is $C_{\varphi(i)}$ with every index j in $br_{\varphi(i)}$ replaced by $\varphi(j)$ for $1 \leq i \leq m$.*

Let I be an interpretation of $\mathcal{KB}$, $M^{\mathsf{S}(I)} = \left\langle C_0^{\mathsf{S}(I)}, C_1^{\mathsf{S}(I)}, \ldots, C_n^{\mathsf{S}(I)} \right\rangle$ be the $\mathsf{S}(I)$-reduct of $\mathsf{M}(\mathcal{KB})$, and $\mathcal{KB}' = \langle \Sigma, sP_\Sigma^I \rangle$ be the strong dl-transform of $\mathcal{P}$ relative to Σ and I, with $M' = \mathsf{M}(\mathcal{KB}') = \langle C'_0, C'_1, \ldots, C'_m \rangle$ the corresponding multi-context system.

Proposition 2 *M' is a subsystem of $M^{\mathsf{S}(I)}$; also, it is a definite context.*

Proof. To see that M' is a subsystem of $M^{\mathsf{S}(I)}$, we first characterize M'. Since the set of input contexts χ'_i in sP_Σ^I is, in general, a subset of the χ_j in $\mathcal{P}$ (the removal of some dl-atoms and some dl-rules may have caused some input contexts to cease to occur), the indices in $\mathsf{M}(\mathcal{KB})$ and M' will not coincide. Let $\varphi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ be the appropriate renaming function, i.e. such that $\chi'_i = \chi_{\varphi(i)}$.

1. C'_0 is $C_0^{\mathsf{S}(I)}$, with every index $i \neq 0$ in br_0 replaced by $\varphi(i)$.

   This can be seen by observing that the rules removed from $\mathcal{P}$ in the construction of sP_Σ^I correspond precisely to the rules removed from C_0 in the construction of $C_0^{\mathsf{S}(I)}$. Consider a ground rule r obtained from grounding a rule in $\mathcal{P}$. If r does not contain dl-atoms, then $r \in sP_\Sigma^I$ iff $r \in (\mathcal{P}^-)^I = \mathsf{red}_0(kb_0, I)$.

   Suppose that r contains dl-atoms. Then r will not be included in sP_Σ^I if r contains a negative literal l such that $I \not\models l$. There are two cases: if l is a regular literal $\neg p$, this means simply that $I \models p$, hence $p \in S_0^I = I$; if l is $\neg DL[\chi_i; Q](t)$, then by Lemma 1 $Q(t) \in S_i^I$. In either case, the corresponding bridge rule will be removed from br_0. This reasoning is reversible, so the converse implication also holds. If those conditions do not hold, then r is included in sP_Σ^I by removing its negative literals – and since $\sigma_{\mathcal{KB}}$ transforms negative literals into negative literals, the bridge rule obtained is the same as removing all negative literals from the bridge rule derived from r.

2. $C'_i = C_{\varphi(i)}^{\mathsf{S}(I)}$.

   The only non-trivial part of this equality regards the bridge rules. But all bridge rules in C'_i are of the form $(\neg)P \leftarrow (0 : p)$, which do not contain negations in their body, so they are never removed.

This shows that M' is a subsystem of $M^{\mathsf{S}(I)}$. The fact that it is a definite multi-context system is a straightforward consequence of the definition of sP^I_Σ. $\Box$

From an interpretation J of $\mathcal{KB}$, we can generate belief states for M and M'; to distinguish them, we will write the subscripts in S explicitly.

Proposition 3 *For any interpretation J of $\mathcal{KB}$, $\mathsf{S}_{\mathcal{KB}}(J)$ is a minimal equilibrium of $M^{\mathsf{S}_{\mathcal{KB}}(I)}$ iff $\mathsf{S}_{\mathcal{KB}'}(J)$ is a minimal equilibrium of M'.*

Proof. The direct implication is straightforward.

For the converse implication, note that from an equilibrium $S = \langle S_0, S_1, \ldots, S_n\rangle$ of $M^{\mathsf{S}_{\mathcal{KB}}(I)}$ we can construct an equilibrium $S' = \langle S_0, S_{\varphi(1)}, \ldots, S_{\varphi(m)}\rangle$ of M'. By minimality of $\mathsf{S}_{\mathcal{KB}'}(J)$, $J = {S'}^J_0 \subseteq S_0$; therefore, if $S_0 \subseteq S^J_0 = J$, we would again have $S_0 = J$, whence $S = \mathsf{S}_{\mathcal{KB}}(J)$. $\Box$

Corollary 2 *Let $S = \langle S_0, S_1, \ldots, S_n\rangle$ be a minimal equilibrium of $M^{\mathsf{S}_{\mathcal{KB}(I)}}$. Then $S = \mathsf{S}_{\mathcal{KB}}(S_0)$.*

Proof. Suppose $S = \langle S_0, S_1, \ldots, S_n\rangle$ is a minimal equilibrium of $M^{\mathsf{S}_{\mathcal{KB}(I)}}$. From S, we can construct a minimal equilibrium $S' = \left\langle S_0, S_{\varphi(1)}, \ldots, S_{\varphi(m)}\right\rangle$ as in Proposition 3, which is a minimal equilibrium of M'. By Corollary 1, $S' = \mathsf{S}_{\mathcal{KB}'}(S_0)$, whence by Proposition 3, it follows that $\mathsf{S}_{\mathcal{KB}}(S_0)$ is also a minimal equilibrium of $M^{\mathsf{S}_{\mathcal{KB}(I)}}$. But $M^{\mathsf{S}_{\mathcal{KB}(I)}}$ is a definite multi-context system, so it only has one minimal equililibrium. Hence $S = \mathsf{S}_{\mathcal{KB}}(S_0)$. $\Box$

Theorem 3 *I is a strong answer set for $\mathcal{KB}$ iff $\mathsf{S}(I)$ is a grounded equilibrium of $\mathsf{M}(\mathcal{KB})$.*

Proof. I is a strong answer set for $\mathcal{KB}$ iff I is the least model of $\mathcal{KB}'$ iff $\mathsf{S}_{\mathcal{KB}'}(I)$ is a minimal equilibrium of M' iff $\mathsf{S}_{\mathcal{KB}}(I)$ is a minimal equilibrium of $M^{\mathsf{S}_{\mathcal{KB}}(I)}$ iff $\mathsf{S}_{\mathcal{KB}}(I)$ is a grounded equilibrium of $\mathsf{M}(\mathcal{KB})$. $\Box$

Both well-founded semantics for dl-programs and well-founded equilibria for multi-context systems are defined in terms of an anti-monotonic operator that computes the minimal model of the adequate generalization of the Gelfond–Lifschitz transform of its argument. We now make this correspondence precise.

Proposition 4 *For every interpretation I of $\mathcal{KB}$, $\gamma_M(\mathsf{S}(I)) = \mathsf{S}\left(\gamma_{\mathcal{KB}}(I)\right)$.*

Proof. Let I be an interpretation of $\mathcal{KB}$. By definition, $\gamma_M(\mathsf{S}(I))$ is the minimal equilibrium of $M^{\mathsf{S}(I)}$, the reduct of M relative to $\mathsf{S}(I)$, hence $\gamma_M(\mathsf{S}(I))$ may be written as $\mathsf{S}_{\mathcal{KB}}(J)$ for some J. By Proposition 3, $\mathsf{S}_{\mathcal{KB}'}(J)$ is a minimal equilibrium of $\mathsf{M}(\mathcal{KB}')$, with $\mathcal{KB}' = \langle \Sigma, sP^I_\Sigma\rangle$, and by Theorem 2 J is the least model of $\mathcal{KB}'$, hence $J = \gamma_{\mathcal{KB}}(I)$ as we wanted to show. $\Box$

Theorem 4 *I is the well-founded semantics of $\mathcal{KB}$ iff $\mathsf{S}(I)$ is the well-founded equilibrium of $\mathsf{M}(\mathcal{KB})$.*

Proof. The proof of this result amounts to showing that $\mathsf{lfp}\left(\gamma^2_M\right) = \mathsf{S}(\mathsf{lfp}\left(\gamma^2_{\mathcal{KB}}\right))$.

1. For every interpretation I of $\mathcal{KB}$ and every $n > 0$,
$$\gamma_M^n(\mathsf{S}(I)) = \mathsf{S}\left(\gamma_{\mathcal{KB}}^n(I)\right) .$$
For $n = 1$ this is simply Proposition 4. Assume the equality holds for n; then
$$\gamma_M^{n+1}(\mathsf{S}(I)) = \gamma_M\left(\gamma_M^n(\mathsf{S}(I))\right) = \gamma_M\left(\mathsf{S}\left(\gamma_{\mathcal{KB}}^n(I)\right)\right) = \mathsf{S}\left(\gamma_{\mathcal{KB}}\left(\gamma_{\mathcal{KB}}^n(I)\right)\right) = \mathsf{S}\left(\gamma_{\mathcal{KB}}^{n+1}(I)\right) ,$$
where the second equality holds by induction hypothesis and the third by Lemma 4.

2. Let $\bot = \langle \emptyset, \emptyset, \ldots, \emptyset \rangle$ be the least belief state of M. Note that it is not true, in general, that $\mathsf{S}(\emptyset) = \bot$, hence we cannot invoke the previous result.

   However, $\bot \sqsubseteq \mathsf{S}(\emptyset)$, where $\sqsubseteq$ is pointwise set inclusion. By monotonicity of γ_M^2, it follows that $\gamma_M^{2n}(\bot) \sqsubseteq \gamma_M^{2n}(\mathsf{S}(\emptyset)) = \mathsf{S}\left(\gamma_{\mathcal{KB}}^{2n}(\emptyset)\right)$ by the previous result. Therefore, $\mathsf{lfp}\left(\gamma_M^2\right) \sqsubseteq \mathsf{S}\left(\mathsf{lfp}\left(\gamma_{\mathcal{KB}}^2\right)\right)$.

   From Corollary 2, we can write $\mathsf{lfp}\left(\gamma_M^2\right)$ as $\mathsf{S}(J)$ for some interpretation J, since γ_M is defined as the minimal equilibrium of a reduct of M. It follows that $\mathsf{S}(J) = \gamma_M^2(\mathsf{S}(J)) = \mathsf{S}\left(\gamma_{\mathcal{KB}}^2(J)\right)$, and hence J is a fixpoint of $\gamma_{\mathcal{KB}}^2$ (since S is injective). Therefore $\mathsf{lfp}\left(\gamma_{\mathcal{KB}}^2\right) \sqsubseteq J$, and by monotonicity of S it follows that $\mathsf{S}\left(\mathsf{lfp}\left(\gamma_{\mathcal{KB}}^2\right)\right) \sqsubseteq \mathsf{S}(J) = \mathsf{lfp}\left(\gamma_M^2\right)$. $\square$

5 Generalizations and applications

We now look at generalizations and applications of the results presented above. We extend the mechanism for generating a multi-context system from a dl-program to multi-dl-programs and their syntactic extensions [5], and show how this mechanism allows constructions originally proposed for (multi-)dl-programs to be automatically ported to MCSs by means of a concrete example.

5.1 Multi-dl-programs

Multi-dl-programs, Mdl-programs for short, were introduced in [5] as a natural extension of dl-programs where several distinct description logic knowledge bases are available – an idea that had already been tackled e.g. in [6]. Formally, an Mdl-program is a pair $\langle \{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P} \rangle$, where each Σ_i is a description logic knowledge base and $\mathcal{P}$ is a logic program whose rules can contain multi-dl-atoms (Mdl-atoms, for short). An Mdl-atom is simply a dl-atom annotated by an index that indicates the knowledge base the query should be addressed to: the semantics of e.g. $DL_3[\chi; Q](t)$ is defined as the semantics of $DL[\chi; Q](t)$ in a traditional dl-program with knowledge base Σ_3.

An Mdl-program $\mathcal{KB} = \langle \{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P} \rangle$ generates a multi-context system $\mathsf{M}(\mathcal{KB})$ in a similar way as a dl-program. For each $i = 1, \ldots, m$, we consider the set $\chi_1^i, \ldots, \chi_{n_i}^i$ of input contexts in Mdl-atoms querying Σ_i. The translation $\sigma_{\mathcal{KB}}$ of literals and Mdl-atoms now requires a sequential enumeration ψ of the input contexts in $\mathcal{P}$.[5] If L is a

[5]In other words, ψ maps (i, j) to the position χ_j^i appears in the sequence of all input contexts in $\mathcal{P}$.

literal, then $\sigma_{\mathcal{KB}}(L) = (0 : L)$; for Mdl-atoms, $\sigma_{\mathcal{KB}}(DL_i[\chi^i_j; Q](t)) = (\psi(i,j) : Q(t))$; and $\sigma_{\mathcal{KB}}(\mathbf{not}\, DL_i[\chi^i_j; Q](t)) = \mathbf{not}(\psi(i,j) : Q(t))$. The translation of $\mathcal{P}$ and each Σ_i is then the same as for dl-programs (using this new $\sigma_{\mathcal{KB}}$), and the multi-context system generated by $\mathcal{KB}$ contains all these contexts.

The semantics for Mdl-programs are defined in exactly the same way as for dl-programs. Again, an interpretation I for $\mathcal{KB}$ generates a belief state $\mathsf{S}_{\mathcal{KB}}(I)$ of $\mathsf{M}(\mathcal{KB})$ by taking $S^I_0 = I$ and $S^I_{\psi(i,j)}$ to be

$$ACC(\Sigma_i \cup \{P(t) \mid I \models p(t), P \uplus p \in \chi^i_j\} \cup \{\neg P(t) \mid I \models p(t), P \mathbin{\cup\!\!\!\!-} p \in \chi^i_j\}) \,,$$

and the following results are straightforward.

Theorem 5 *Let $\mathcal{KB} = \langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}\rangle$ be an Mdl-program and I be an interpretation for $\mathcal{KB}$.*

1. *If I is a model of $\mathcal{KB}$, then $\mathsf{S}(I)$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$.*
2. *If $S = \langle S_0, \ldots, S_n\rangle$ is an equilibrium of $\mathsf{M}(\mathcal{KB})$, then S_0 is a model of $\mathcal{KB}$.*
3. *If $\mathcal{KB}$ is positive, then I is the least model of $\mathcal{KB}$ iff $\mathsf{S}(I)$ is a minimal equilibrium of $\mathsf{M}(\mathcal{KB})$.*
4. *I is a strong answer set for $\mathcal{KB}$ iff $\mathsf{S}(I)$ is a grounded equilibrium of $\mathsf{M}(\mathcal{KB})$.*
5. *I is the well-founded semantics of $\mathcal{KB}$ iff $\mathsf{S}(I)$ is the well-founded equilibrium of $\mathsf{M}(\mathcal{KB})$.*

5.2 Extensions to Mdl-programs

Often when working with an Mdl-program, one wishes to observe predicates from one component in another, e.g. to have a predicate p in $\mathcal{P}$ such that $p(t)$ holds whenever Σ_i proves $S(t)$ for some concept S. This is achieved by adding the rule $p(X) \leftarrow DL_i[; S](X)$ (or the binary equivalent, if S is a role) to $\mathcal{P}$.

Reciprocally, one may want to observe p (or $\neg p$) in Σ_i using a concept or role S. In order to achieve this, one has to add $S \uplus p$ (or $S \mathbin{\cup\!\!\!\!-} p$) to *every* Mdl-atom querying Σ_i in $\mathcal{KB}$. This poses consistency issues during the development of an Mdl-program: if at some stage one decides that S is an observer of p, one has to remember to add $S \uplus p$ (or $S \mathbin{\cup\!\!\!\!-} p$) to every *future* Mdl-atom querying Σ_i. It is a well-known fact in the programming community that this kind of constraint is one of the major sources of errors in software design.

In order to encode this kind of global changes, the authors introduced *observers* in [5]. An observer is a pair (S, p) or (p, S), where S is a concept or role from Σ_i and p is a predicate of the same arity from $\mathcal{P}$, with intended meaning that the second component of the pair should include all instances of the first. An *Mdl-program with observers* is then $\langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}, \{\Lambda_1, \ldots, \Lambda_m\}, \{\Psi_1, \ldots, \Psi_m\}\rangle$, where Λ_i and Ψ_i collect the observers from and to each Σ_i. In turn, from an Mdl-program with observers we obtain an Mdl-program $\left\langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}^{\Psi_1,\ldots,\Psi_m}_{\Lambda_1,\ldots,\Lambda_m}\right\rangle$ where $\mathcal{P}^{\Psi_1,\ldots,\Psi_m}_{\Lambda_1,\ldots,\Lambda_m}$ is obtained from $\mathcal{P}$ by:

- adding the rule $p(X) \leftarrow DL_i[;S](X)$ for each $\langle S, p\rangle \in \Lambda_i$;
- in each dl-atom $DL_i[\chi; Q](t)$ (including those added in the previous step), adding $S \uplus p$ to χ for each $\langle p, S\rangle \in \Psi_i$ and $S \cup\!\!\!\!-\, p$ to χ for each $\langle p, \neg S\rangle \in \Psi_i$.

We have extended `dlvhex` with the capability of processing several ontologies, and thereby computing answer sets of Mdl-programs. Furthermore, a pre-processor module was added to the DL-plugin, allowing the user to write Mdl-programs with observers that are expanded into (plain) Mdl-programs as described above.

Clearly, we can translate an Mdl-program with observers to a multi-context system in two steps. More interestingly, we can also define a *direct* translation as follows.

Definition 4 *The multi-context system $\mathsf{M}(\mathcal{KB})$ generated by the Mdl-program with observers $\mathcal{KB} = \langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}, \{\Lambda_1, \ldots, \Lambda_m\}, \{\Psi_1, \ldots, \Psi_m\}\rangle$ is obtained as follows.*

1. *Construct $M = \mathsf{M}(\langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}\rangle)$.*
2. *If necessary, add contexts to M corresponding to Mdl-atoms with empty input context for each Σ_i. For each i, let this context be C_{i^*}.*[6]
3. *For each $(S, p) \in \Lambda_i$, adding the bridge rule $p \leftarrow (i^* : S)$ to br_0.*
4. *For each $(p, S) \in \Psi_i$, adding the bridge rule $S \leftarrow (0 : p)$ to each set $br_{\psi(i,j)}$, with $j = 1, \ldots, n_i$, and to br_{i^*}.*

This definition captures the intended meaning of the observers, as expressed by the following result.

Theorem 6 *Let $\mathcal{KB} = \langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}, \{\Lambda_1, \ldots, \Lambda_m\}, \{\Psi_1, \ldots, \Psi_m\}\rangle$ be an Mdl-program with observers. Then*

$$\mathsf{M}(\mathcal{KB}) = \mathsf{M}\left(\left\langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}^{\Psi_1,\ldots,\Psi_m}_{\Lambda_1,\ldots,\Lambda_m}\right\rangle\right) .$$

Proof. The construction of $\left\langle\{\Sigma_1, \ldots, \Sigma_m\}, \mathcal{P}^{\Psi_1,\ldots,\Psi_m}_{\Lambda_1,\ldots,\Lambda_m}\right\rangle$ consists of two steps.

The first step adds Mdl-atoms to $\mathcal{P}$ with the empty context, thereby ensuring that this context occurs in $\mathcal{P}$, and adds a rule to $\mathcal{P}$ that is translated to the bridge rule added in step 3 of the construction of $\mathsf{M}(\mathcal{KB})$.

The second step adds $S \uplus p$ to χ^i_j for each $\langle p, S\rangle \in \Psi_i$ and $S \cup\!\!\!\!-\, p$ to χ^i_j for each $\langle p, \neg S\rangle \in \Psi_i$, for every $j = 1, \ldots, n_i$ (and the empty context in Mdl-atoms querying Σ_i, if it was only added in the previous step). When computing the generated multi-context system, these extra input information will yield new bridge rules in $br_{\psi(i,j)}$ and in the context eventually added in the first step; these are precisely the bridge rules added in step 4 of the construction of $\mathsf{M}(\mathcal{KB})$. □

Thus, from an Mdl-program with observers we can obtain a multi-context system without first having to "unfold" the observers into a simple Mdl-program.

[6]If there is an input context $\chi^i_j = \emptyset$ in $\mathcal{P}$, just take i^* to be $\psi(i, j)$.

5.3 Design patterns in multi-context systems

In real life, a substantial amount of the time required in software development is spent in finding and implementing design solutions for recurrent problems already addressed and for which good solutions already exist. For this reason, an important field in research is that of identifying common scenarios and proposing mechanisms to deal with these scenarions – the so-called *design patterns* for software development.

Within this setting, mappings between different formalisms are of the utmost usefulness, since they allow design patterns that were originally conceived in one formalism to be automatically ported to another formalism. The resulting design patterns can often be optimized – but it is also common knowledge that optimizing a known solution is usually much simpler than coming up with an efficient solution from scratch.

Several design patterns for Mdl-programs were proposed in [5], and we will present a few of these and show how they can be translated to design patterns in MCSs. In most cases, this will correspond to a more general degree of applicability, since the latter are a more general framework. In a few cases, we will look at some trivial ways to optimize the resulting design pattern.

One of the simplest design patterns is the ***Observer*** pattern, applicable when there is a predicate in $\mathcal{P}$ that should include all instances of a concept or role S in some Σ_i (or reciprocally). This pattern is implemented in an Mdl-program with observers simply by adding the pair (S, p) (or (p, S)) to the appropriate observer set Λ_i (or Ψ_i). As discussed above, this corresponds to adding a bridge rule $p \leftarrow (i^* : S)$ to br_0 (or $S \leftarrow (0 : p)$ to all contexts generated from Σ_i).

This mechanism makes sense in a generic MCS, i.e. in one that is not generated from an Mdl-program. Therefore, we immediately obtain a more general ***Observer*** design pattern that we can apply in any multi-context system whenever we want to ensure that some S_i in context C_i is updated every time another S_j in context C_j is changed: simply add the rule $S_i \leftarrow (j : S_j)$ to br_i.

Note that this implementation is necessarily vague since there are no restrictions on the logics underlying each context. Therefore, S_i and S_j may be predicate symbols from a logic program, or concepts or roles from a description logic, as in a multi-context system generated from an Mdl-program, but they may also be propositional variables from a propositional logic. Typically, however, the usage of the name "logic" suggests that they should be some kind of predicate symbol; for simplicity, we will assume throughout that this is the case.

A more interesting example arises when one looks at the ***Polymorphic Entities*** design pattern. The setting for this pattern is the following: in $\mathcal{P}$, there is a predicate p whose instances are inherited from other $Q_1, \ldots, Q_k$ where each Q_j is a concept or role from $\Sigma_{\varphi(j)}$. Again, in Mdl-programs this pattern is implemented by adding a number of observers, namely (Q_j, p) to each $\Lambda_{\varphi(j)}$. In the generated multi-context system, this corresponds to adding bridge rules $p \leftarrow (\varphi(j)^* : Q_j)$ for each j to br_0.

Once again, we can generalize this pattern to a generic MCS, whenever we want predicate P from context C_i to inherit all instances of predicates $Q_1, \ldots, Q_k$ where each Q_j is a predicate from a context $C_{\varphi(j)}$. This is achieved by adding bridge rules $P \leftarrow (\varphi(j) : Q_j)$ for each j to br_i. Note that, even for an MCS generated from an Mdl-program, this is a more general construction, since it also allows a concept or role from a description logic knowledge base to be affected by concepts or roles in other knowledge bases or by predicates in $\mathcal{P}$.

An example where we can simplify the design pattern we obtain is when we want to add closed world semantics to a predicate in one of the contexts. In the setting of Mdl-programs, where each description logic knowledge base has open-world semantics and the logic program has default negation, this is achieved by the ***Closed World*** design pattern. To give closed-world semantics to a concept (or role) S in Σ_i, we choose fresh predicate symbols s^+ and s^- in $\mathcal{P}$ and add (S, s^+) to Λ_i, $(s^-, \neg S)$ to Ψ_i and the rule $s^-(X) \leftarrow \textbf{not}\, s^+(X)$ to $\mathcal{P}$.

In the generated MCS, this corresponds to adding $s^+ \leftarrow (i^* : S)$ to br_0, $\neg S \leftarrow (0 : s^-)$ to br_{i^*}, and the rule $s^-(X) \leftarrow \textbf{not}\, s^+(X)$ to kb_0. Generalizing this to an arbitrary MCS is not immediate, however, since there is not necessarily a component where default negation is available. On the other hand, default negation *is* an ingredient of bridge rules, so we can forego the use of the auxiliary predicates s^+ and s^- and simply give closed-world semantics to a predicate S in context C_i by adding the bridge rule $\neg S \leftarrow \textbf{not}(i : S)$ to br_i.

The last design pattern we discuss here is ***Adapter***, which is applied whenever a component Σ_k of a system is not known or available at the time of implementation of others, yet it is necessary to query it. In an Mdl-program, one would add an empty interface knowledge base Σ_I whose language includes the desired concept and role names, and later connect each concept and role in Σ_i with its counterpart in Σ_k by means of observers through $\mathcal{P}$.

One can again simplify this pattern when in the setting of general MCSs, since it is possible to connect each predicate from C_I with the corresponding predicate (not necessarily of the same name) from C_k by means of bridge rules in br_I. Furthermore, this pattern does not suffer from the limitations of the original ***Straight Adapter*** pattern in Mdl-programs, since these limitations were strictly related to the existence of local queries, which are not present in MCSs.

It is also interesting to notice that this last pattern can be modified in a very simple way to implement a proxy: simply add side-conditions to the body of the bridge rules connecting C_I with C_k that restrict the communication between these two contexts.

6 Conclusions

The basic constructs of dl-programs and multi-context systems are based upon different motivations, and are therefore fundamentally different. In this paper, we showed how, even so, an arbitrary dl-program can be translated into an MCS, which is equivalent to it in a very precise way – namely, the interpretations of the dl-program naturally give rise to belief states for the generated MCS, and this correspondence matches specific classes of semantic structures. Thus, models become equilibria, minimal models become minimal equilibria, answer sets become grounded equilibria, and well-founded semantics (for dl-programs) become well-founded semantics (for MCSs).

An important aspect of this construction is that we can *compute* minimal equilibria and well-founded semantics for MCSs generated from dl-programs, which is not true in general (the definition of minimal equilibrium is a characterization that is not computational: minimal equilibria cannot be constructed in a systematic way). Also, there is an algorithmic procedure to check that an equilibrium for an MCS generated from a dl-program is grounded, which again is not true in general.

Finally, we showed how this technique can be applied to generalize some useful constructions on dl-programs to similar constructions in (general) MCSs.

References

[1] 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.

[2] G. Brewka and T. Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 385–390. AAAI Press, 2007.

[3] G. Brewka, T. Eiter, and M. Fink. Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In M. Balduccini and T.C. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 233–258. Springer–Verlag, 2011.

[4] L. Cruz-Filipe, P. Engrácia, G. Gaspar, and I. Nunes. Achieving tightness in dl-programs. Technical Report 2012;03, Faculty of Sciences of the University of Lisbon, July 2012. Available at `http://hdl.handle.net/10455/6872`.

[5] L. Cruz-Filipe, I. Nunes, and G. Gaspar. Patterns for programming in the semantic web. Technical Report 2012;06, Faculty of Sciences of the University of Lisbon, October 2012. Available at `http://www.di.fc.ul.pt/~in/papers/TR-2012-06.pdf`.

[6] 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.

[7] The `dlvhex` tool. `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.