Description logics, rules and multi-context systems

Luís Cruz-Filipe[1,2,4], Rita Henriques[3], and Isabel Nunes[3,4]

[1] Escola Superior Náutica Infante D. Henrique, Portugal
[2] CMAF, Lisboa, Portugal
[3] Faculdade de Ciências, Universidade de Lisboa
[4] LabMag, Lisboa, Portugal

Abstract. The combination of rules and ontologies 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 formalisms, Mdl-programs and multi-context systems, which address different aspects of this combination, and include different, incomparable programming constructs. Despite this, we show that every Mdl-program can be transformed in a multi-context system, and this transformation relates the different semantics for each paradigm in a natural way. As an application, we show how a set of design patterns for multi-context systems can be obtained from previous work on Mdl-programs.

1 Introduction

Several approaches combining rules and ontologies have been proposed in the last years for semantic web reasoning, e.g. [2, 8–10, 12, 16] among others. Ontologies are typically expressed through decidable fragments of function-free first-order logic with equality, offering a very good ratio of 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 non-monotonic 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: Mdl-programs [7], which are a straightforward generalization of the well-known 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 – an Mdl-program is essentially a logic program that can query description logic knowledge bases, "feeding" its view of the latter with newly inferred facts; MCSs, on the other hand, consist of several knowledge bases, possibly expressed in different languages, each declaring additional rules that allow communication with the others.

Despite their differences, we show that every Mdl-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, any useful constructions developed within the framework of Mdl-programs may be automatically translated to equivalent constructions in the setting of MCSs. Although the idea behind the syntactic translation is suggested in [3] to justify that MCSs generalize the original dl-programs, even this claim is not substantiated beyond an intuitive perspective. Here, we will not only make this syntactic correspondence precise, but discuss in detail the semantic correspondences it entails, and apply it to obtain a set of design patterns for MCSs based on such a set for Mdl-programs.

The structure of the paper is as follows. Section 2 recalls the syntax and semantics of Mdl-programs. Section 3 introduces the syntax of MCSs and the translation of Mdl-programs into these. Section 4 summarizes the different semantics of MCSs and relates the semantics of an Mdl-program and those of the MCS it generates. Section 5 applies this correspondence to design patterns for Mdl-programs. Section 6 concludes the paper.

2 Mdl-programs

This section presents multi-description logic programs [7], Mdl-programs for short, which are a straightforward generalization of dl-programs, an already established framework for coupling description logic knowledge bases with rule-based reasoning [9, 10]. The main advantage of Mdl-programs, as we will see, is their simplicity. Although they do not possess the level of generality other systems such as HEX-programs [11] or multi-context systems [2] have, Mdl-programs are quite adequate for reasoning within the semantic web, where a lot of effort is being put into developing ontologies, which for the main part are description logic knowledge bases.

2.1 Syntax

The purpose of Mdl-programs is to generalize logic programs with special atoms that communicate with external description logic knowledge bases, which we will refer to henceforth simply as "knowledge bases". The key ingredient of Mdl-programs is the notion of *dl-atom*. A dl-atom relative to a set of knowledge bases $\overline{\Sigma} = \{\Sigma_1, \ldots, \Sigma_n\}$ and a function-free first-order signature Φ is $DL_i\,[S_1\ op_1\,p_1, \ldots, S_m\ op_m\,p_m; Q]\,(\bar{t})$, where $1 \leq i \leq n$, each S_k is either a concept or role of Σ_i, or a special symbol in $\{=, \neq\}$; $op_i \in \{\uplus, \sqcup\}$; each p_k is a unary or binary predicate symbol of Φ, according to whether the corresponding S_k is a concept or a role; and $Q(\bar{t})$ 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 from Σ_i, R is a role from Σ_i, and t, t_1 and t_2 are terms – constants from any Σ_j or Φ, or variables. 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.

Note that no requirement is made about any relations between the different knowledge bases; in particular, the description logics underlying the Σ_is need not be the same.

A *dl-rule* over $\overline{\Sigma}$ and Φ is a Horn clause that may contain dl-atoms, i.e. it has the form $a \leftarrow b_1, \ldots, b_k, \mathbf{not}\ b_{k+1}, \ldots, \mathbf{not}\ b_m$ where a is a logical atom and $b_1, \ldots, b_m$ are either logical atoms or dl-atoms – where the logical atoms are again built using terms from Φ and constants from any Σ_i. The *head* of r is a and the *body* of r is $b_1, \ldots, b_k, \mathbf{not}\ b_{k+1}, \ldots, \mathbf{not}\ b_m$. An *Mdl-program* over Φ is a pair $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ where $\overline{\Sigma}$ is as before, each Σ_i is a description logic knowledge base and $\mathcal{P}$ is a finite set of dl-rules over $\overline{\Sigma}$ and Φ (also referred to as a *generalized logic program*). As usual, we will omit referring to Φ explicitly. Note that negation in $\mathcal{P}$ is the usual, closed-world, negation-as-failure, in contrast with the knowledge bases Σ_i, which (being description logic knowledge bases) come with an open-world semantics. An Mdl-program with only one knowledge base ($n = 1$) is a *dl-program* – and this definition coincides with that in [9].

The operators $\uplus$ and $\cup$ extend Σ_i in the context of the current dl-query. Intuitively, $S_k \uplus p_k$ (resp., $S_k \cup p_k$) adds to S_k (resp., $\neg S_k$) all instances of (pairs of) terms for which p_k holds – the *extent* of p_k –, before evaluating the query. This only affects $\mathcal{P}$'s current view of Σ_i without changing Σ_i. The components of an Mdl-program are thus kept independent, communicating only through dl-atoms; so, although they function separately, giving Mdl-programs nice modularity properties, there is a bidirectional flow of information via dl-atoms.[1]

We will adopt some notational conventions throughout this paper. Variables are capital letters in math font (e.g. X), while constants and terms are in sans serif. Predicate symbols (from the generalized logic program) begin with a lowercase letter, while concepts and roles (from the knowledge base) are written exactly as they are defined in the source ontologies. We will *not* use different fonts for objects of $\mathcal{P}$ and objects of the Σ_is, since these sets are not necessarily disjoint (the constants of all Σ_is may be used in $\mathcal{P}$); we will however abstain from using the same name for a predicate in $\mathcal{P}$ and a concept or role in Σ_i.

Example 1. We illustrate the syntax of Mdl-programs with a simple example. This program uses two external knowledge bases: Σ_1 is the Travel Ontology `travel.owl` [13], freely available online, which defines a series of travel-related concepts, including that of (tourist) Destination; and Σ_2 is a freely accessible wine ontology `wine.rdf` [15], which compiles a substantial amount of information about wines, including the locations of several important wineries around the world; in particular, this ontology contains a concept Region identifying some major wine regions throughout the world. These are combined in an Mdl-program by means of the following generalized logic program $\mathcal{P}$.

$$\mathsf{wineDest}(\mathsf{Tasmania}) \leftarrow \qquad (r_1)$$

$$\mathsf{wineDest}(\mathsf{Sydney}) \leftarrow \qquad (r_2)$$

$$\mathsf{wineDest}(X) \leftarrow DL_2[;\mathsf{Region}](X) \qquad (r_3)$$

$$\mathsf{overnight}(X) \leftarrow DL_1[;\mathsf{hasAccommodation}](X, Y) \qquad (r_4)$$

$$\mathsf{oneDayTrip}(X) \leftarrow DL_1[\mathsf{Destination} \uplus \mathsf{wineDest};\mathsf{Destination}](X), \mathbf{not}\ \mathsf{overnight}(X) \qquad (r_5)$$

[1] The original definition of dl-programs included a third operator, but it can be defined as an abbreviation of the other two [9], and we follow this methodology.

This program defines a predicate wineDest with two instances, corresponding to two wine regions that are interesting tourist destinations, and a rule (r_3) extending the definition of wineDest with a query to Σ_2, importing all instances of Region in Σ_2. Informally, the goal is that wineDest should be a new subconcept of Destination, but Σ_1 is left unchanged.

Rules (r_4, r_5) identify the destinations only suitable for one-day trips. The possible destinations are selected via (r_5) not only from the information originally in Σ_1, but by (i) extending the concept Destination of Σ_1 with all instances of wineDest in $\mathcal{P}$ and then (ii) querying this extended view of Σ_1 for all instances of Destination. The result is then filtered using the auxiliary predicate overnight defined in (r_4) as the set of destinations for which some accommodation is known. This uses the role hasAccommodation of Σ_1, where hasAccommodation(t_1,t_2) holds whenever t_1 is a Destination and t_2 an accommodation facility located in t_1. The reason for resorting to (r_4) at all is the usual one in logic programming: the operational semantics of negation-as-failure requires all variables in a negated atom to appear in non-negated atoms in the body of the same rule.[2]

An interesting aspect of this example is that Sydney is already an individual of Σ_1 – one of the characteristics of Mdl-programs is precisely that the atoms of $\mathcal{P}$ may use constants of Σ_1 as terms. This is relevant: rule (r_1) adds new constants to the domain of $\mathcal{KB}$, but rule (r_2) adds *information* about an individual already in Σ_1. Note the relevance of the extended query in (r_5): if Destination were not updated with the information from wineDest, we would not be able to infer e.g. oneDayTrip(Tasmania). In the next section we will introduce semantics for dl-programs and show that this is indeed a consequence of $\mathcal{KB}$.

2.2 Semantics

In order to provide semantics for Mdl-programs, we first recall the notion of Herbrand base of a logic program $\mathcal{P}$ over Φ, denoted $\mathsf{HB}_{\mathcal{P}}$ – the set of all ground atoms consisting of predicate symbols and terms from Φ. The Herbrand base of an Mdl-program $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$, denoted $\mathsf{HB}_{\mathcal{KB}}$, is similarly defined, except that constant symbols may also come from the vocabulary of the Σ_is. An *interpretation* is a subset I of $\mathsf{HB}_{\mathcal{KB}}$. We say that I is a *model* of a ground atom or dl-atom a under $\overline{\Sigma}$, or I *satisfies* a under $\overline{\Sigma}$, denoted $I \models_{\overline{\Sigma}} a$, in the following cases:

- if $a \in \mathsf{HB}_{\mathcal{KB}}$, then $I \models_{\overline{\Sigma}} a$ iff $a \in I$;
- if a is a ground dl-atom $DL_i[\chi; Q](t)$ with $\chi = S_1\, op_1\, p_1, \ldots, S_m\, op_m\, p_m$, then $I \models_{\overline{\Sigma}} a$ iff $\Sigma_i(I;\chi) \models Q(\overline{t})$, where $\Sigma_i(I;\chi) = \Sigma_i \cup \bigcup_m^{j=1} A_j(I)$ and, for $1 \leq j \leq m$,

$$A_j(I) = \begin{cases} \{S_j(\overline{e}) \mid p_j(\overline{e}) \in I\}, & op_j = \uplus \\ \{\neg S_j(\overline{e}) \mid p_j(\overline{e}) \in I\}, & op_j = \cup \end{cases}$$

[2] We do not need to extend Destination in this dl-rule because of the structure of Σ_1: the role hasAccommodation is defined as the set of its instances (without any axioms), so changing other concepts or roles has no effect on its semantics.

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

Example 2. Given the dl-program $\mathcal{KB}$ of Example 1, its Herbrand base contains all ground atoms built from applying wineDest, overnight and oneDayTrip not only to the constants of $\mathcal{P}$ – Tasmania and Sydney – but also to all individuals of Σ_1, which include (among others), Canberra and FourSeasons (which is not an instance of Destination), and of Σ_2, which includes e.g. AustralianRegion. Thus, $\mathsf{HB}_{\mathcal{KB}}$ contains e.g.

wineDest(AustralianRegion)	overnight(Tasmania)	oneDayTrip(Canberra)
wineDest(FourSeasons)	overnight(Sydney)	oneDayTrip(Sydney), ...

This may seem a bit strange, since e.g. wineDest(FourSeasons) does not fit well with our intended interpretation of wineDest; but this is a well-known side-effect of the absence of types in logic programming.

This program has only one model. To analyze it, one has to know that the only instance of hasAccommodation in Σ_1 has Sydney as its first argument. Thus, this model contains overnight(Sydney), as well as wineDest(Tasmania) and wineDest(Sydney); furthermore, it includes wineDest(t) for every t such that $\Sigma_2 \models \mathsf{Region}(t)$. Finally, for every individual t other than Sydney such that $\Sigma_1 \models \mathsf{Destination}(t)$ or $\Sigma_2 \models \mathsf{Region}(t)$, the model contains oneDayTrip(t).

This model may not seem like a very realistic view of the world, but this is a limitation of the current state of the underlying ontologies.

An Mdl-program $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ is *positive* if the rules in $\mathcal{P}$ do not contain negations. Positive Mdl-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 the Σ_is to evaluate dl-atoms. The Mdl-program in Example 1 is not a positive program because of the negation in rule (r_4).

Answer-set semantics. The answer set semantics of (not necessarily positive) Mdl-programs is defined again in analogy to that of logic programs. Given an Mdl-program $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$, we can obtain a positive dl-program by replacing $\mathcal{P}$ with its *dl-transform* $s\mathcal{P}^I_{\overline{\Sigma}}$ relative to $\overline{\Sigma}$ and an interpretation I. This is obtained by grounding every rule in $\mathcal{P}$ and then (i) deleting every dl-rule r such that $I \models_{\overline{\Sigma}} a$ 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 $\mathcal{KB}^I = \langle \overline{\Sigma}, s\mathcal{P}^I_{\overline{\Sigma}} \rangle$ is a positive Mdl-program, it has a unique least model $\mathsf{M}_{\mathcal{KB}^I}$. An *answer set* of $\mathcal{KB}$ is an interpretation I that coincides with $\mathsf{M}_{\mathcal{KB}^I}$.

The model of the Mdl-program in the previous example is also an answer set of that program.

Well-founded semantics. Another semantics for Mdl-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 an Mdl-program $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ 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 earlier. This operator is anti-monotonic, so $\gamma^2_{\mathcal{KB}}$ is monotonic and therefore it has a least and greatest fixpoint, denoted $\mathsf{lfp}\left(\gamma^2_{\mathcal{KB}}\right)$ and $\mathsf{gfp}\left(\gamma^2_{\mathcal{KB}}\right)$, respectively. An atom $a \in \mathsf{HB}_{\mathcal{P}}$ is *well-founded* if $a \in \mathsf{lfp}\left(\gamma^2_{\mathcal{KB}}\right)$ and *unfounded* if $a \notin \mathsf{gfp}\left(\gamma^2_{\mathcal{KB}}\right)$; 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 $\mathcal{KB}$.

The well-founded semantics of the previous example contains all atoms in its models and the negations of all remaining atoms. This is a consequence of there being only one answer set for that Mdl-program (see [9] for details).

2.3 Mdl-programs with observers

On top of Mdl-programs, we defined a syntactic construction [7] to extend a concept or role from one of the Σ_is (in $\mathcal{P}$'s view of Σ_i) with all instances of a predicate in $\mathcal{P}$, or reciprocally. An *Mdl-program with observers* is a pair $\langle \mathcal{KB}, \mathcal{O} \rangle$, where $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ is an Mdl-program, $\mathcal{O} = \langle \{\Lambda_1, \ldots, \Lambda_n\}, \{\Psi_1, \ldots, \Psi_n\} \rangle$, the observer sets, where Λ_i is a finite set of pairs $\langle S, p \rangle$ and Ψ_i is a finite set of pairs $\langle p, S \rangle$, in both cases with S a (negated) concept from Σ_i and p a unary predicate from $\mathcal{P}$, or S a (negated) role from Σ_i and p a binary predicate from $\mathcal{P}$.

Intuitively, Λ_i contains the concepts and roles in Σ_i that $\mathcal{P}$ needs to observe, in the sense that $\mathcal{P}$ should be able to detect whenever new facts about them are derived, whereas Ψ_i contains the predicates in $\mathcal{P}$ that Σ_i needs to observe. Note that a specific symbol (be it a predicate, concept or role) may occur in different Ψ_is or Λ_is. An Mdl-program with observers can be transformed in a (standard) Mdl-program by replacing $\mathcal{P}$ with $\mathcal{P}^{\mathcal{O}}$, obtained from $\mathcal{P}$ by:

- adding rule $p(X) \leftarrow DL_i[;S](X)$ for each $\langle S, p \rangle \in \Lambda_i$, if S is a concept (and its binary counterpart, if S is a role); and
- in each dl-atom $DL_i[\chi; Q](\bar{t})$ (including those added in the previous step), adding $S \uplus p$ to χ for each $\langle p, S \rangle \in \Psi_i$ and $S \uplus p$ to χ for each $\langle p, \neg S \rangle \in \Psi_i$.

Example 3. We can rewrite the previous Mdl-program as an Mdl-program with observers by omitting rule (r_3) and taking $\Lambda_2 = \{\langle \mathsf{Region}, \mathsf{wineDest} \rangle\}$.

Having in mind the structure of Σ_1 (see footnote on page 4), we can go a step further, take $\Psi_1 = \{\langle \mathsf{wineDest}, \mathsf{Destination} \rangle\}$ and replace (r_5) with

$$\mathsf{oneDayTrip}(X) \leftarrow DL_1[;\mathsf{Destination}](X), \mathbf{not}\ \mathsf{overnight}(X) \qquad (r'_5)$$

Unfolding this observer now yields a program with the same semantics but where rule (r_4) has been replaced by

$$\mathsf{overnight}(X) \leftarrow DL_1[\mathsf{Destination} \uplus \mathsf{wineDest}; \mathsf{hasAccommodation}](X, Y) \quad (r'_4)$$

3 From Mdl-programs to multi-context systems

There are other approaches to combining different reasoning and programming paradigms; in particular, rules and ontologies can also be combined within AL-log [8], HEX-programs [11], MKNF [14] and multi-context systems [2, 4]. Some of these systems even allow for more general combinations; however, Mdl-programs, being less general, are easier to manipulate and understand.

In this section, we show that Mdl-programs (with observers) can be translated to MCSs. The converse is trivially not true – MCSs are far more general, as their definition shows. Besides being an interesting result by itself, this translation will be used in Section 5 to guide the development of an elementary set of design patterns for MCSs – another useful contribution, since the latter framework is more general than most other existing approaches [4].

Instead of presenting MCSs on their own, this section is organized as follows. We begin by defining their syntax and immediately follow with the definition of the (syntactic) translation from Mdl-programs to MCSs. Then, we introduce the several semantics for MCSs together with the correspondence results that relate the semantics of an Mdl-program and the corresponding MCS. In this way, the constructions and results are more easily appreciated.

Multi-context systems were originally proposed [2] as a way of combining different reasoning paradigms, where information flows among the different logics within the system through bridge rules.

The notion of multi-context system is defined in several layers.

1. A *logic* is as a triple $L = (KB_L, BS_L, ACC_L)$ where KB_L is the set of well-formed knowledge bases of L; BS_L is the set of possible belief sets; and $ACC_L : KB_L \to 2^{BS_L}$ is a function describing the semantics of the logic by assigning to each element of KB_L 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 L, 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_L and BS_L is defining these notions for each logic L.
2. Given a set of logics $\{L_1, \ldots, L_n\}$, an L_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 $1 \leq r_i \leq n$; each p_i is an element of some belief set of L_{r_i}; and $kb \cup \{s\} \in KB_k$ for each $kb \in KB_k$.
3. A *multi-context system* (MCS) $M = \langle C_1, \ldots, C_n \rangle$ is a collection of contexts $C_i = (L_i, kb_i, br_i)$, where $L_i = (KB_i, BS_i, ACC_i)$ is a logic, $kb_i \in KB_i$ is a knowledge base, and br_i is a set of L_i-bridge rules over $\{L_1, \ldots, L_n\}$.

Given an Mdl-program $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$, there are two steps in the process of generating an MCS from $\mathcal{KB}$.

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 idea behind this syntactic construction is suggested in [4], it is not defined precisely, neither are the semantic implications discussed. Here, we will formalize this syntactic correspondence and analyze its implications at the semantic level.

Definition 1. *Let $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ be an Mdl-program. For each $i = 1, \ldots, n$, let $\chi^i_1, \ldots, \chi^i_{m_i}$ be the input contexts in dl-atoms querying Σ_i. Let ψ be a sequential enumeration of all input contexts in $\mathcal{P}$, i.e. $\psi(i,j)$ is the position of χ^i_j in the sequence of all input contexts in $\mathcal{P}$.*

1. *The translation $\sigma_{\mathcal{KB}}$ of literals and dl-atoms is defined by*

$$\sigma_{\mathcal{KB}}(a) = \begin{cases} (0 : L) & \text{if } a \text{ is a literal } L \\ (\psi(i,j) : Q(t)) & \text{if } a = DL_i[\chi^i_j; Q](t) \\ \mathbf{not}(\psi(i,j) : Q(t)) & \text{if } a = \mathbf{not}\, DL_i[\chi^i_j; Q](t) \end{cases}$$

2. *The translation of $\mathcal{P}$ is the context $C_0 = \langle L_0, kb_0, br_0 \rangle$ where:*
   - *$L_0 = \langle KB_0, BS_0, ACC_0 \rangle$ 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 $\mathcal{P} \setminus \mathcal{P}^-$.*
3. *For each input context $\chi^i_j = P_1\, op_1\, p_1, \ldots, P_k\, op_k\, p_k$, the context $C_{\psi(i,j)} = \langle L_i, kb_i, br_{\psi(i,j)} \rangle$ is defined as follows.*
   - *$L_i = \langle KB_i, BS_i, ACC_i \rangle$ is the description logic underlying Σ_i, with KB_i the set of all knowledge bases over Σ_i's signature; BS_i contains all sets of dl-queries to Σ_i; and ACC_i assigns to each knowledge base the set of dl-queries it satisfies.*[3]
   - *kb_i is Σ_i.*
   - *For $j = 1, \ldots, k$, $br_{\psi(i,j)}$ 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_i and kb_i are the same for all contexts originating from Σ_i.
4. *The MCS* generated *by $\mathcal{KB}$, $\mathsf{M}(\mathcal{KB})$, contains C_0 and all the $C_{\psi(i,j)}$.*

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.

[3] Formally, we can define ACC_i as computing the set of logical consequences of the knowledge base and restricting it to those formulas that are dl-queries.

Example 4. Recall the Mdl-program $\mathcal{KB}$ from Example 1. For the purpose of generating an MCS from $\mathcal{KB}$, observe that there are two different input contexts associated with Σ_1, $\chi_1 = \epsilon$ and $\chi_2 = \mathsf{Destination} \uplus \mathsf{wineDest}$, and one associated with Σ_2, $\chi_3 = \epsilon$. Rules (r_1) and (r_2) do not include dl-atoms, so they belong to $\mathcal{P}^-$. Rules (r_3), (r_4) and (r_5), which contain dl-atoms, are translated as the following L_0-bridge rules.

$$\mathsf{wineDest}(X) \leftarrow (3 : \mathsf{Region}(X)) \qquad (r'_3)$$
$$\mathsf{overnight}(X) \leftarrow (1 : \mathsf{hasAccommodation}(X, Y)) \qquad (r'_4)$$
$$\mathsf{oneDayTrip}(X) \leftarrow (2 : \mathsf{Destination}(X)), \mathbf{not}\ \mathsf{overnight}(X) \qquad (r'_5)$$

The generated multi-context system $\mathsf{M}(\mathcal{KB})$ is thus $\langle C_0, C_1, C_2, C_3 \rangle$, where:

$$C_0 = \langle L_0, \{r_1, r_2\}, \{r'_3, r'_4, r'_5\} \rangle \qquad C_1 = \langle L_1, \Sigma_1, \emptyset \rangle$$
$$C_2 = \langle L_1, \Sigma_1, \{\mathsf{Destination}(X) \leftarrow (0 : \mathsf{wineDest}(X))\} \rangle \qquad C_3 = \langle L_2, \Sigma_2, \emptyset \rangle$$

Note that, formally, the syntax of MCSs does not allow variables, so $\mathsf{M}(\mathcal{KB})$ should instead include the ground versions of these rules. However, it is usual to write MCSs with variables for readability and succinctness [4].

An interesting aspect is that we can translate an Mdl-program with observers *directly* to an MCS (without first "unfolding" the observers) as follows.

Definition 2. *Let $\langle \mathcal{KB}, \mathcal{O} \rangle$ be an Mdl-program with observers. The MCS it generates is $\mathsf{M}(\mathcal{KB}, \mathcal{O})$, defined as follows.*

1. *Construct $M = \mathsf{M}(\mathcal{KB})$.*
2. *Without loss of generality, assume that M contains contexts C_{i^*} corresponding to the empty input context for each Σ_i.*
3. *For each $(S, p) \in \Lambda_i$, add the bridge rule $p \leftarrow (i^* : S)$ to br_0.*
4. *For each $(p, S) \in \Psi_i$, add the bridge rule $S \leftarrow (0 : p)$ to each $br_{\psi(i,j)}$, with $j = 1, \ldots, n_i$, and to br_{i^*}.*

This construction captures the intended meaning of the observers.[4]

Theorem 1. *Let $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ be an Mdl-program and $\mathcal{O}$ be observer sets for $\mathcal{KB}$. Then $\mathsf{M}(\mathcal{KB}, \mathcal{O}) = \mathsf{M}\left(\langle \overline{\Sigma}, \mathcal{P}^{\mathcal{O}} \rangle\right)$.*

4 Semantics of multi-context systems

There are several different semantics for MCSs, all of which are defined in terms of the semantics for the logics in the individual contexts.

Let $M = \langle C_1, \ldots, C_n \rangle$ be a multi-context system, with $C_i = (L_i, kb_i, br_i)$ for each $1 \leq i \leq n$. A *belief state* for M is a collection $S = \langle S_1, \ldots, S_n \rangle$ of belief sets for each context, i.e. $S_i \in BS_i$ for each $1 \leq i \leq n$.

[4] The proofs of all results can be found in [6].

A bridge rule $s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \mathbf{not}(r_{j+1} : p_{j+1}), \ldots, \mathbf{not}(r_m : p_m)$ is *applicable* in belief state $S = \langle S_1, \ldots, S_n \rangle$ iff $p_i \in S_{r_i}$ for $1 \leq i \leq j$ and $p_k \notin S_{r_k}$ for $j+1 \leq k \leq m$. A belief state $S = \langle S_1, \ldots, S_n \rangle$ of M is an *equilibrium* if the condition $S_i \in ACC_i(kb_i \cup \{H(r) \mid r \in br_i \text{ is applicable in } S\})$ holds for $1 \leq i \leq n$, where $H(r)$ denotes the head of rule r as usual.

In other words, belief states are simply "candidate" models, in the sense that they are acceptable as potential models of each context. Information is transported between different contexts by means of bridge rules, since a bridge rule in one context may refer to other contexts, and the notion of equilibrium guarantees that all belief states are not only models of the local information at each context, but also reflect the relationships imposed by the bridge rules.

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

Definition 3. *Let $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ be an Mdl-program and I be an interpretation of $\mathcal{KB}$. The belief state* generated *by I is $\mathsf{S}_{\mathcal{KB}}(I) = \langle S_0^I, S_1^I, \ldots, S_m^I \rangle$ of $\mathsf{M}(\mathcal{KB})$, where $S_0^I = I$ and $S_{\psi(i,j)}^I$ is the only element of*

$$ACC_i\left(\Sigma_i \cup \{P(t) \mid I \models p(t), P \uplus p \in \chi_j^i\} \cup \{\neg P(t) \mid I \models p(t), P \uplus\ p \in \chi_j^i\}\right).$$

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 Mdl-program under consideration, we omit the subscript in $\mathsf{S}_{\mathcal{KB}}$.

In the example from the previous section, one can check that its model generates a belief state that is also an equilibrium of $\mathsf{M}(\mathcal{KB})$. This suggests that there are very close connections between I and $\mathsf{S}(I)$, which we now prove formally.

Theorem 2. *Let $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ be an Mdl-program.*

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

Furthermore, since ACC_i always yields a singleton set, equilibria for MCSs generated from an Mdl-program can be uniquely derived from their first component, as expressed by the following corollary.

Corollary 1. *If $S = \langle S_0, \ldots, S_m \rangle$ 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 $\mathcal{P}$.

Minimal equilibria. As is the case in logic programming, there can be too many equilibria for a given multi-context system; for this reason, several particular kinds of equilibria are defined in [2], reflecting different kinds of preferences one may adopt to choose among them. These categories closely follow the usual hierarchy for models of logic programs, as well as of Mdl-programs. The basic concept is that of minimal equilibrium. This is a relative notion, since (as discussed in [2]) it may not make sense to minimize the belief sets for *all* contexts.

Let $M = \langle C_1, \ldots, C_n \rangle$ be a multi-context system and $C^* \subseteq \{C_1, \ldots, C_n\}$ be the set of contexts of M whose models should be minimized. An equilibrium

$S = \langle S_1, \ldots, S_n \rangle$ of M is C^*-*minimal* if there is no equilibrium $S' = \langle S'_1, \ldots, S'_n \rangle$ of M such that: (1) $S'_i \subseteq S_i$ for all $C_i \in C^*$; (2) $S'_i \subsetneq S_i$ for some $C_i \in C^*$; and (3) $S'_i = S_i$ for all $C_i \notin C^*$. In this paper, we will always use $C^* = M$ and simply refer to *minimal* equilibria, which the reader should understand to mean M-minimal equilibria.

Since the transformation S from interpretations of Mdl-programs to belief states preserves inclusions, we also have the following relationship.

Theorem 3. *Let $\mathcal{KB} = \langle \overline{\Sigma}, \mathcal{P} \rangle$ be an Mdl-program. Then I is the least model of $\mathcal{KB}$ iff $\mathsf{S}(I)$ is a minimal equilibrium of $\mathsf{M}(\mathcal{KB})$.*

Minimal equilibria (or even C^*-equilibria) do not necessarily exist. In logic programming, it is shown that least models always exist for positive programs, a result that holds also for dl-programs [9] and Mdl-programs. In MCSs, this class corresponds to that of definite multi-context systems.

A logic L is *monotonic* if $ACC_L(kb)$ is always a singleton set, and $kb \subseteq kb'$ implies that the only element of $ACC_L(kb)$ is a subset of the only element of $ACC_L(kb')$. This coincides with the usual notion of monotonic logic. A logic $L = (KB_L, BS_L, ACC_L)$ is *reducible* if (1) there is $KB^*_L \subseteq KB_L$ such that the restriction of L to KB^*_L is monotonic; and (2) there is a reduction function $\mathsf{red}L : KB_L \times BS_L \to KB^*_L$ such that, for each $k \in KB_L$ and $S, S' \in BS_L$, (2a) $\mathsf{red}_L(k, S) = k$ whenever $k \in KB^*_L$; (2b) $\mathsf{red}L$ is anti-monotonic in the second argument; and (2c) $S \in ACC_L(k)$ iff $ACC_L(\mathsf{red}_L(k, S)) = \{S\}$. A context $C = (L, kb, br)$ is *reducible* if (1) L is reducible; and (2) for all $H \subseteq \{H(r) \mid r \in br\}$ and belief sets S, $\mathsf{red}_L(kb \cup H, S) = \mathsf{red}_L(kb, S) \cup H$. A multi-context system is *reducible* if all of its contexts are reducible.

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, $kb_i = \mathsf{red}_{L_i}(kb_i, S)$ for all i and every $S \in BS_i$). Every definite MCS has a unique minimal equilibrium [2], which we will denote by $Eq(M)$.

Grounded equilibria. The semantics of non-definite MCSs is defined via a generalization of the Gelfond–Lifschitz reduct to the multi-context case. If $M = \langle C_1, \ldots, C_n \rangle$ is a reducible MCS and $S = \langle S_1, \ldots, S_n \rangle$ is a belief state of M, then the *S-reduct* of M is $M^S = \langle C_1^S, \ldots, C_n^S \rangle$, where $C_i^S = (L_i, \mathsf{red}_{L_i}(kb_i, S_i), br_i^S)$ and, for each i, br_i^S is obtained from br_i by (1) deleting every rule with some **not** $(k : p)$ in the body such that $p \in S_k$, and (2) deleting all **not** literals from the bodies of remaining rules. If $S = Eq(M^S)$, then S is a *grounded equilibrium* of M.

Note that this definition only makes sense if M^S is definite; indeed, it has been shown [2] that this is always the case. In particular, 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.

Answer sets for Mdl-programs correspond to grounded equilibria for MCSs. This should not come as a big surprise: both the dl-transform of Mdl-programs

and the reduct of an MCS are generalizations of the Gelfond–Lifschitz transform of ordinary logic programs.

Theorem 4. *I is an answer set for $\mathcal{KB}$ iff $\mathsf{S}(I)$ is a grounded equilibrium of $\mathsf{M}(\mathcal{KB})$.*

Well-founded semantics. The well-founded semantics for reducible MCSs is also defined in [2], based on the operator $\gamma_M(S) = Eq(M^S)$. Since γ_M is anti-monotonic, γ_M^2 is monotonic as usual. However, one can only guarantee the existence of its least fixpoint by the Knaster–Tarski theorem if BS_i has a least element for each logic Σ_i in any of M's contexts. If this is the case, then the well-founded semantics of M is $WFS(M) = \mathsf{lfp}\left(\gamma_M^2\right)$.

As with models of logic programs (and of Mdl-programs), $WFS(M)$ is not necessarily an equilibrium: it contains the knowledge that is common to all equilibria, but being an equilibrium is not preserved by intersection.[5]

This definition is very similar to that of well-founded semantics for Mdl-programs. Therefore, the following result should not come as a surprise.

Theorem 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 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 scenarios – the so-called *design patterns* for software development. With this in mind, the authors proposed an elementary set of design patterns for Mdl-programs [7].

In this section, we apply the translation from Mdl-programs to MCSs to obtain an initial set of design patterns for MCSs, and discuss how adequate the resulting patterns are. This discussion focuses on the potential usefulness of the translation: we capitalize on the mapping previously defined to port interesting constructions from one formalism to another automatically. As it turns out, we can overcome some of the problems that affected the set described in [7], thereby obtaining a more expressive set of design patterns for MCSs. Also, we take advantage of the intrinsic structure of MCSs to optimize some of the resulting design patterns, a simpler process than developing efficient ones from scratch.

The simplest design pattern is the ***Observer Down*** 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. This pattern is implemented in an Mdl-program with observers simply by adding the pair (S, p) to the appropriate observer set Λ_i. According

[5] In view of the following result, to obtain an example, pick a dl-program whose well-founded semantics is not a model (see [5]) and apply the translation defined above.

to Definition 2, this corresponds to adding a bridge rule $p \leftarrow (i^* : S)$ to br_0. Reciprocally, the pattern ***Observer Up*** allows S to include all instances of p, and is implemented by adding $S \leftarrow (0 : p)$ to all contexts generated from Σ_i.

Looking at these constructions from the perspective of MCSs, their implementation follows the same structure: a bridge rule with exactly one literal in its body is added to a context C_i, thereby updating some S_i in C_i's language using input from some S_j in C_j. This mechanism makes sense in general, regardless of whether the MCS is generated from an Mdl-program or not. We thus obtain a general ***Observer*** design pattern that we can apply in any MCS 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.

This pattern also captures the ***Transversal Observer*** pattern of Mdl-programs, applicable when one knowledge base needs to observe another; however, this implementation is simpler than translating this pattern directly, since in MCSs there is in general no need to use an intermediate context. This type of simplification will also be used in other patterns.

A more interesting example arises when one looks at the ***Polymorphic Entities*** design pattern. The setting is the following: in $\mathcal{P}$, there is a predicate p whose instances are inherited from $Q_1, \ldots, Q_k$, where each Q_j is a concept or role from Σ_j. This pattern is again implemented by adding a number of observers, namely (Q_j, p) to each Λ_j. In the generated MCS, this corresponds to adding bridge rules $p \leftarrow (j^* : Q_j)$ for each j to br_0. This pattern can be applied in 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_j. This is achieved by adding bridge rules $P \leftarrow (j : Q_j)$ for each j to br_i.

An example where we can substantially simplify the design pattern obtained is adding closed world semantics to a predicate in some context. 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 $s^-(X) \leftarrow \mathbf{not}\, s^+(X)$ 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 $s^-(X) \leftarrow \mathbf{not}\, s^+(X)$ to kb_0.

To generalize this pattern, we first observe that adding $s^-(X) \leftarrow \mathbf{not}\, s^+(X)$ to kb_0 is equivalent to adding the bridge rule $s^- \leftarrow \mathbf{not}(0 : s^+)$ to br_0, since the semantics of bridge rules is that of logic programs. As before, the context C_0 is now being used solely as an intermediate for a construction that can be made directly in C_i; therefore, we can implement ***Closed-World*** in an MCS, giving closed-world semantics to a predicate S_i in context C_i by adding the bridge rule $\neg S_i \leftarrow \mathbf{not}(i : S_i)$ to br_i. Once again, this pattern makes sense in any MCS, regardless of the nature of its components – as long as the context C_i has negation.

The last design pattern we discuss here is ***Adapter***, which is applied whenever a component Σ_k is not known or available at the time of implementation of

others, yet it is necessary to query it. In an Mdl-program, one adds 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 ***Transversal Observer***. This pattern works without any changes in any MCS; however, the resulting program will be simpler because the application of ***Observer*** yields a simpler MCS than the application of ***Transversal Observer*** in an Mdl-program.

Furthermore, as was observed in [7], in Mdl-programs this pattern does not work well if one needs dl-atoms querying Σ_I which locally extend this knowledge base. In MCSs, this problem does not arise, and thus this implementation of ***Adapter*** is closer to the spirit of this pattern in e.g. object-oriented programming. It is also interesting to notice that this 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. As was observed in [7], it is not clear whether a proxy can be implemented in Mdl-programs.

The ideas in this section constitute an initial approach to the study of design patterns in multi-context systems. We point out that we obtained for free a set of design patterns including all design patterns for Mdl-programs in [7], applicable in a more general setting. Furthermore, several of these patterns were simplified in a systematic way, removing indirections resulting from the need, in Mdl-programs, to go through the logic program in order to establish communication between two different knowledge bases.

6 Conclusions

The basic constructs of Mdl-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 Mdl-program can be translated into an MCS, which is equivalent to it in a very precise way – namely, the interpretations of the Mdl-program naturally give rise to belief states for the generated MCS, taking this correspondence to the semantic level. Thus, (minimal) models become (minimal) equilibria, answer sets become grounded equilibria, and well-founded semantics (for Mdl-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 Mdl-programs efficiently, which is not true in general (the definition of minimal equilibrium is a characterization that is not computational, and minimal equilibria cannot usually be constructed in a practical way, except by brute-force testing of all candidates). Also, there is an algorithmic procedure to check whether an equilibrium for an MCS generated from a Mdl-program is grounded, which again is not true in general.

Finally, we showed how this technique can be applied to obtain a start set of design patterns for MCSs. This set was obtained by translating a pre-existing set of design patterns for Mdl-programs and simplifying the result following some

general principles motivated by the specificities of MCSs. In some cases, the resulting patterns turned out to be more encompassing than the original ones. We intend to use this work as a first step in a more comprehensive study of design patterns for multi-context systems.

Acknowledgments

The authors wish to thank Graça Gaspar for the fruitful discussions and her insightful comments on a first version of the manuscript.

This work was partially supported by Fundação para a Ciência e Tecnologia under contract PEst-OE/EEI/UI0434/2011.

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 *AAAI2007*, 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 *LNCS*, pages 233–258. Springer, 2011.
4. G. Brewka, T. Eiter, M. Fink, and A. Weinzierl. Managed multi-context systems. In T. Walsh, editor, *IJCAI*, pages 786–791. IJCAI/AAAI, 2011.
5. 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`.
6. L. Cruz-Filipe, R. Henriques, and I. Nunes. Viewing dl-programs as multi-context systems. Technical Report 2013;05, Faculty of Sciences of the University of Lisbon, April 2013. Available at `http://hdl.handle.net/10455/6895`.
7. L. Cruz-Filipe, I. Nunes, and G. Gaspar. Patterns for interfacing between logic programs and multiple ontologies. To appear in *Proceedings of KEOD'2013*; available at `http://tinyurl.com/itsweb2013-09`, 2013.
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 Nr 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. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In L.P. Kaelbling and A. Saffiotti, editors, *IJCAI2005*, pages 90–96. Professional Book Center, 2005.

12. 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, *ECAI2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 35–40. IOS Press, 2010.
13. H. Knublauch. Travel ontology 1.0. Available at `http://protege.cim3.net/file/pub/ontologies/travel/travel.owl`.
14. B. Motik and R. Rosati. Reconciling description logics and rules. *Journal of the ACM*, 57, June 2010. Article Nr 30.
15. The OWL Working Group. Wine ontology. Available at `http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf`.
16. R. Rosati. DL+log: Tight integration of description logics and disjunctive Datalog. In P. Doherty, J. Mylopoulos, and C.A. Welty, editors, *KR2006*, pages 67–78. AAAI Press, 2006.

A Proof from Section 3

Proof (Theorem 1). The construction of $\langle \overline{\Sigma}, \mathcal{P}^{\mathcal{O}} \rangle$ consists of two steps.

The first step adds dl-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 $\mathsf{M}(\mathcal{KB}, \mathcal{O})$.

The second step adds $S \uplus p$ to χ_j^i for each $\langle p, S \rangle \in \Psi_i$ and $S \uplus p$ to χ_j^i for each $\langle p, \neg S \rangle \in \Psi_i$, for every $j = 1, \ldots, n_i$ (and the empty context in dl-atoms querying Σ_i, if it was only added in the previous step). When computing the generated MCS, this 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 $\mathsf{M}(\mathcal{KB}, \mathcal{O})$. □

B Proofs from Section 4

For legibility, we will prove all results assuming there is only one description logic, i.e. $\mathcal{KB} = \langle \Sigma, \mathcal{P} \rangle$ is a dl-program. This allows us to choose $\psi(1, i) = i$ and significantly simplifies notation. The generalization to Mdl-programs is straightforward, since each knowledge base is processed independently.

Theorem 2

This result relies on the following lemma.

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 mimics 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. □

Proof (Theorem 2).

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 is the only element of $ACC(\Sigma \cup \{P(t) \mid I \models p(t), P \uplus p \in \chi_i\} \cup \{\neg P(t) \mid I \models p(t), P \uplus p \in \chi_i\})$, 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 $\mathcal{KB}$. □

In case 1 of the previous proof, nothing is said about the case when a bridge rule is not applicable: in this situation, the definition of equilibrium poses no restrictions on whether or not its head is part of the corresponding belief set.

Theorem 3

Proof.

(⇒) 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 2, 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.

(⇐) 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 $\mathcal{P}$. □

Theorem 4

This proof proceeds in several steps. Throughout this proof, 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. □

The dl-transform of $\mathcal{P}$ generates a subsystem of the reduction of $\mathsf{M}(\mathcal{KB})$ in the following sense.

Definition 4. *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 $1 \leq i \leq m$.*

In particular, if $M' \subseteq M$, then $m \leq n$.

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 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^I_Σ 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 $p \in S^I_0 = I$; 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^I_Σ 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. $C'_i = C^{\mathsf{S}(I)}_{\varphi(i)}$.
   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 MCS is a straightforward consequence of the definition of sP^I_Σ. □

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 $S = \mathsf{S}_{\mathcal{KB}}(J)$. □

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 $S' = \langle S_0, S_{\varphi(1)}, \ldots, S_{\varphi(m)}\rangle$ 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 equilibrium. Hence $S = \mathsf{S}_{\mathcal{KB}}(S_0)$. □

Proof (Theorem 4). I is an 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 $\mathsf{M}(\mathcal{KB})$. □

Theorem 5

This proof relies on the following result.

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 3 J is the least model of $\mathcal{KB}'$, hence $J = \gamma_{\mathcal{KB}}(I)$ as we wanted to show. □

Proof (Theorem 5). 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 $n > 0$,

   $$\gamma^n_M(\mathsf{S}(I)) = \mathsf{S}\left(\gamma^n_{\mathcal{KB}}(I)\right) .$$

   For $n = 1$ this is simply Proposition 4. Assume the equality holds for n; then

   $$\begin{aligned}\gamma^{n+1}_M(\mathsf{S}(I)) &= \gamma_M\left(\gamma^n_M(\mathsf{S}(I))\right) = \gamma_M\left(\mathsf{S}\left(\gamma^n_{\mathcal{KB}}(I)\right)\right) \\ &= \mathsf{S}\left(\gamma_{\mathcal{KB}}\left(\gamma^n_{\mathcal{KB}}(I)\right)\right) = \mathsf{S}\left(\gamma^{n+1}_{\mathcal{KB}}(I)\right) ,\end{aligned}$$

   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^2_M, it follows that $\gamma^{2n}_M(\bot) \sqsubseteq \gamma^{2n}_M(\mathsf{S}(\emptyset)) = \mathsf{S}\left(\gamma^{2n}_{\mathcal{KB}}(\emptyset)\right)$ by the previous result. Therefore, $\mathsf{lfp}\left(\gamma^2_M\right) \sqsubseteq \mathsf{S}\left(\mathsf{lfp}\left(\gamma^2_{\mathcal{KB}}\right)\right)$.
   From Corollary 2, we can write $\mathsf{lfp}\left(\gamma^2_M\right)$ 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 $\mathsf{S}(J) = \gamma^2_M(\mathsf{S}(J)) = \mathsf{S}\left(\gamma^2_{\mathcal{KB}}(J)\right)$, and hence J is a fixpoint of $\gamma^2_{\mathcal{KB}}$ (since S is injective). Therefore $\mathsf{lfp}\left(\gamma^2_{\mathcal{KB}}\right) \sqsubseteq J$, and by monotonicity of S it follows that $\mathsf{S}\left(\mathsf{lfp}\left(\gamma^2_{\mathcal{KB}}\right)\right) \sqsubseteq \mathsf{S}(J) = \mathsf{lfp}\left(\gamma^2_M\right)$. □