Dynamic Logic Programming: Various Semantics Are Equal on Acyclic Programs

Martin Homola

Comenius University, Bratislava, Slovakia,
homola@tbc.sk

Abstract. Multidimensional dynamic logic programs (MDLPs) are suitable to represent knowledge dynamic in time, or more generally, information coming from various sources, partially ordered by arbitrary relevancy relation, e.g., level of authority. They have been shown useful for modeling and reasoning about multi-agent systems. Various approaches to define semantics of MDLPs have been presented. Most of the approaches can be characterized as based on rejection of rules.
It is understood that on some restricted classes of MDLPs several of these semantics coincide. We focus on acyclic programs. We show that for a MDLP $\mathcal{P}$ and a candidate model M, if $\mathcal{P}$ is acyclic to some extent then several of the known semantics coincide on M. It follows as a direct consequence that on the class of acyclic programs all of these semantics coincide.

1 Introduction

Background. In *Multidimensional Dynamic Logic Programs* (*MDLPs*), introduced in [1], knowledge is encoded into several logic programs, partially ordered by a relevance relation. MDLPs have been shown as well suited for representing knowledge change in time, and as well, to provide favourable representation for reasoning over information structured by some relevancy relation, such as authority hierarchies.

Various approaches have been presented in order to provide semantics of MDLPs. Most of the approaches utilize rejection of rules. Such semantics include $\mathcal{P}$-Justified Update semantics introduced in [2, 3], Dynamic Stable Model semantics from [4, 1], Update Answer Set semantics from [5, 3] and Refined Dynamic Stable Model semantics of [6, 7]. (The latest one is only known for linearly ordered MDLPs.) Typically, semantics assigns a set of models to a program. Models are picked among the interpretations of the program. Usually, a new semantics has been introduced to cope with drawbacks of the older ones.

Most important contributions are those of Leite [3, 8], Eiter et al. [5, 9] and Alferes et al. [4, 6]. Authors point out that for some particular pairs of semantics, for a given MDLP, the model-set of one semantics is always a subset of the model-set of the other one. Thus, a sort of hierarchy of the model-sets assigned to a MDLP by different semantics is organized (cf. [3, 5, 6, 10]).

Also, it is a shared opinion, that on "plain" MDLPs, which are not obfuscated with cyclic dependencies (cyclic chains of rules), conflicting rules within a same logic program and other unconvenient constructs, all of these semantics coincide. Different behavior on some "abnormal" MDLPs is usually assigned to the inability of some of the semantics to deal with these abnormalities. Several restrictive conditions on MDLPs have been introduced in order to identify classes of programs on which two or more semantics coincide (cf. [3, 5, 6]).

Relationship with multi-agent systems. Already in [1], authors have shown that MDLPs are useful to model and reason about multi-agent systems. Particularly in logic based multi-agent systems where knowledge of an agent is naturally represented by rules. Thus, knowledge associated with an agent at a given state is encoded into a logic program. Assume that the agent's knowledge evolves with time. With each new time-state new knowledge appears to the agent, in form of rules, perceived trough sensors or communicated with other agents. This new knowledge may be in general contrary to the knowledge inherited from the previous time-states. We want the agent to be able to resolve such conflicts, assigning more relevance to the more recent knowledge.

MDLPs allow us to do this in a natural way. Agent's initial state and subsequent perceptions are modeled as a sequence of logic programs. More recent information is treated as more relevant. MDLPs assign semantics to the sequence, resolving conflicts between rules according to their relevancy. Moreover, they enable for determining semantics of the agent's knowledge at arbitrary state, thus allowing us to query the agent's knowledge history.

Besides time, MDLPs are capable of handling other relevancy relations, like specificity of the information or authority. This is particularly handy in multi-agent communities where an authoritative hierarchy among the agents is present. Assume that the knowledge of each agent is represented by a logic program. If an agent is authoritatively superior to the another one, we treat also the program of the former one as more relevant than the program of the latter one. Assuming that the agents obey the authority, we are able to query the global knowledge of the system but as well the knowledge of a subsystem rendered by an agent together with all the agents that are inferior to it.

Moreover, the framework allows us to combine several "relevancy dimensions" into a single MDLP. Thus, we are able to model, e.g., the knowledge distributed over an authority-enabled community of agents and as well the change of the whole system in time. Hence, we favor MDLPs as a powerful framework for modeling and reasoning about knowledge distributed over multi-agent systems, logic-based in particular. However, a multi-agent system does not have to be associated with a single MDLP, nor the view provided by the MDLP has to be global. For instance, each agent may use a MDLP to maintain its own view of the system, reflecting its own preference amongst the chunks of information obtained by communication with other agents. Thus, MDLPs may also provide a local knowledge repository for each agent of the system. For a more detailed analysis, we refer the reader to [3, 1, 11, 12]. We also refer the reader to [13], in

order to see how extensions of MDLPs can benefit to multi-agent systems, and to [14] to see how the knowledge of multiple agents can be combined when there is no authoritative order among the agents.

Motivation. At presence, whole family of semantics is available for MDLPs. Most of these semantics, being based on same notions (e.g., generalization of stable model semantics, employing rejection of rules) are very close, one to another. Studying the differences and similarities between them, helps us to evaluate them w.r.t. the intuitions and principles we want them to obey. Also, maybe we do not really need such a rich family of semantics, indeed if the difference between them shows to be very small. Particularly, within the field of multi-agent systems, it helps us to determine whether or not MDLPs are appropriate for a particular application, and if yes, which semantics to choose.

From this point of view, We find several results of [5, 3, 7], about restricted classes of MDLPs on which some of the semantics coincide, not tight, as many MDLPs on which the semantics also match are beyond the proposed classes.

Presence of cyclic dependencies in MDLPs and its responsibility for distinct behavior of these semantics has been investigated in [5, 3, 7]. It has been remarked that perhaps on MDLPs that do not contain cycles several of the semantics may coincide. We see this hypothesis as valuable, since acyclic programs form a broad subclass and it is known that for some, simpler, applications they are sufficient. So, we suggest further evaluation of these semantics w.r.t. the class of acyclic programs and programs with limited occurrence of cyclic dependencies.

Contribution. As in [3, 15], we build MDLPs over a more general language of generalized extended logic programs that unifies the previous approaches under a common framework, allowing for more elegant comparisons, while keeping the previous approaches as special cases, so the results are propagated.

We introduce a new concept of sufficient acyclicity. Logic program is sufficiently acyclic if each of its literals is supported by at least one acyclic derivation. As the main result we establish a restrictive condition, using the notion of sufficient acyclicity, under which four (five) of the semantics coincide on the given interpretation of the given (linear) MDLP. It trivially follows that on acyclic programs these semantics coincide entirely. This article presents the results of the author's Master's thesis [10] that can be viewed as its extended version.

2 Preliminaries

We first introduce basic concepts from logic programming. Logic programs are build from propositional *atoms*. The set of all atoms is denoted by $\mathcal{A}$. We employ two kinds of negation, *explicit negation* $\neg$ and *default negation not* . Let p be a proposition. By $\neg\, p$ we intuitively mean that (we know that) A is not true. Default negation is sometimes called negation as failure. We use it to express lack of objective evidence: by *not* p we intuitively mean that we have no evidence confirming that p is true.

Objective literal is an atom or an atom preceded by explicit negation (e.g., $A \in \mathcal{A}$ and $\neg A$ are objective literals.) *Default literal* is an objective literal preceded by default negation (e.g., *not* A, *not* $\neg A$ are default literals, $A \in \mathcal{A}$.) Both objective literal and default literal are *literals*. We denote the set of all objective literals by $\mathcal{O}$, the set of all default literals by $\mathcal{D}$ and the set of all literals by $\mathcal{L}$.

Rule is a formula $L \leftarrow L_1, \ldots, L_n$, where $n \geq 0$ and $L, L_1, \ldots, L_n \in \mathcal{L}$. Rule of a form $L \leftarrow$ (i.e., $n = 0$) is called a *fact*. For each rule r of a form $L \leftarrow L_1, \ldots, L_n$ we call the literal L the *head* of r and denote it by $h(r)$ and we call the set $\{L_1, \ldots, L_n\}$ the *body* of r and denote it by $b(r)$.

A set of rules P is called a *generalized extended logic program* (hereafter often just *logic program* or *GELP*). GELPs are the most general logic programs that we use. We favor the approach outlined in [3, 15], where MDLPs are built over GELPs, unifying the previous approaches under a common framework, allowing for more elegant comparisons, while keeping the previously used languages as special cases, so the results are propagated. We also remark, that GELPs enable to properly manipulate three truth values, "something is true", "something is false", and "we do not know", allowing to adequately switch from one to another, what we mark as a desirable feature, once dealing with knowledge updates.

Several other flavours of logic programs do exist. We mention *extended logic programs*, a subclass of GELPs formed by programs that do not contain default literals in heads of rules. *Generalized logic programs* do not allow explicit negation at all, i.e., for each objective literal L, contained in the program, it holds that $L \in \mathcal{A}$, and for each default literal *not* L, contained in the program, it holds that $L \in \mathcal{A}$. Logic program is *definite* if it only contains atoms of $\mathcal{A}$ in the heads, as well as in the bodies of its rules, i.e., definite logic programs do not allow negation at all.

Let P be a GELP. The expanded version of P is the program $\dot{P} = P \cup \{not\ \neg h(r) \leftarrow b(r) \mid r \in P \wedge h(r) \in \mathcal{O}\}$. Two literals $L \in \mathcal{O}$ and *not* L are said to be *conflicting*. Two rules are conflicting if their heads are conflicting literals. We denote this by $L \bowtie L'$ and by $r \bowtie r'$ respectively. For any set of literals S, $S^+ = S \cap \mathcal{O}$ and $S^- = S \cap \mathcal{D}$.

A set of literals that does not contain a pair of conflicting literals is called an *interpretation*. Interpretation is *total* if for each $L \in \mathcal{O}$ it contains L or *not* L. Literal L is *satisfied* in the interpretation I if $L \in I$ and we denote it by $I \vDash L$. Also $I \vDash S$, a set of literals S, if $I \vDash L$ for each $L \in S$. A rule r is satisfied in the interpretation I (denoted by $I \vDash r$) if $I \vDash h(r)$ whenever $I \vDash b(r)$. Let P be a definite logic program. We denote by $least(P)$ the unique *least model* of P that exists, as showed by van Emden and Kowalski in [16].

Most of the semantic approaches in dynamic logic programming build on ideas of the stable model semantics of logic programs that has been introduced by Gelfond and Lifschitz in [17]. According to this semantics a total interpretation M is a stable model of a GELP P if it holds that $M = least(P \cup M^-)$[1].

[1] With an abuse of notation, we commonly treat (sets of) facts as (sets of) rules, and also GELPs as definite programs, considering each negated literal as a new atom.

3 MDLPs and Various Semantics Based on Rejection of Rules

Logic programs have been proven as useful media in the area of knowledge representation. As long as the information we deal with is rather static we face no problem to encode it in form of a logic program. But we reach the barrier very soon, when dealing with information change in time, or when integrating information from several sources with various levels of relevancy.

To deal with this problem, the framework of dynamic logic programming has been introducedi in [4]. In this framework information is encoded into several programs that are linearly ordered into a sequence by their level of relevancy. Such sequences are called dynamic logic programs.

This framework has been further generalized in [1] by allowing logic programs ordered by arbitrary (i.e., also non-linear) partial ordering. Multidimensional dynamic logic programs were born. We formalize the latter approach in Definition 1.

Definition 1. *Let $G = (V, E)$ be a directed acyclic graph with finite set of vertices V. Let $\mathcal{P} = \{P_i \mid i \in V\}$ be a set of logic programs. The pair $(\mathcal{P}, G)$ is a* multidimensional dynamic logic program *or often just* program *or* MDLP*.*

We often use just $\mathcal{P}$ instead of $(\mathcal{P}, G)$ and assume the existence of the corresponding G. The multiset of all rules of the expanded versions $\dot{P}_i$ of the logic programs P_i, $i \in V$ of $\mathcal{P}$ is denoted by $\uplus_{\mathcal{P}}$. Let $i, j \in V$, we denote by $i \prec j$ (and also by $P_i \prec P_j$) if there is a directed path from i to j in G. We denote by $i \preceq j$ (and by $P_i \preceq P_j$) if $i \prec j$ or if $i = j$.

A *dynamic logic program* (*DLP*, *linear* MDLP) is such a MDLP $\mathcal{P}$ whose G is collapsed into a single directed path. So DLPs form a subclass of MDLPs, they are precisely all linearly ordered MDLPs.

Most of the semantic approaches in dynamic logic programming are based on the ideas of stable model semantics of simple logic programs. A set of models is assigned to a program by each of these semantics. Models are picked among the interpretations of the program.

As a MDLP in general may contain conflicting rules, semantics try to resolve these conflicts, when it is possible, according to the relevancy level of the conflicting rules. A common approach is to assign a set of *rejected rules* to a given program $\mathcal{P}$ and a "candidate model" interpretation M. Rejected rules are then subtracted from the union of all rules of $\mathcal{P}$, gaining the residue of $\mathcal{P}$ w.r.t. M. Also the set of *default assumptions* (sometimes just *defaults*) is assigned to $\mathcal{P}$ and M. Defaults are picked among the default literals. A fix-point condition is verified, whether M coincides with the least model of the union of the residue and the default assumptions. If so, then M is a model of P w.r.t. the semantics. A semantics that can be characterized in this manner is said to be *based on rejection of rules* or *rule-rejecting*.

Once we deal with several rule-rejecting semantics, then any difference between them originates in the way how particularly rejection of rules and default

assumptions are implemented in these semantics. Two different kinds of rejection have been used with MDLPs. The original rejection used in [4, 1] keeps each rule intact as long as there is no reason for rejecting it in form of a more relevant rule that is satisfied in the considered interpretation. Formally, the set of rejected rules of $\mathcal{P}$ w.r.t. M is

$$Rej(\mathcal{P}, M) = \{r \in \dot{P}_i \mid (\exists r' \in \dot{P}_j)\ i \prec j, M \vDash b(r'), r \bowtie r'\}\ .$$

In [5], an alternative notion of rejection has been introduced, allowing each rule to reject other rules only if it is not rejected already. Such a set of rejected rules of $\mathcal{P}$ w.r.t. M is formalized as

$$Rej^\star(\mathcal{P}, M) = \{r \in \dot{P}_i \mid (\exists r' \in \dot{P}_j)\ i \prec j, M \vDash b(r'), r \bowtie r', r' \notin Rej^\star(\mathcal{P}, M)\}\ .$$

Originally, in [2], default assumptions have been computed just exactly as in the stable model semantics of logic programs. Formally,

$$Def^\star(\mathcal{P}, M) = M^-\ .$$

Later on, in [4, 1], another approach has been introduced, as the original set of defaults showed to be too broad. We formalize defaults according to this approach as

$$Def(\mathcal{P}, M) = \{not\ L \mid L \in \mathcal{O}, (\nexists r \in \uplus_{\mathcal{P}})\ h(r) = L, M \vDash b(r)\}\ .$$

Combining two implementations of rejection and two of default assumptions immediately leads to four semantics of MDLPs. We define each of them formally in the following.

Definition 2. *A rule-rejecting semantics that uses $Rej(\mathcal{P}, M)$ for rejection and $Def^\star(\mathcal{P}, M)$ for defaults is called* dynamic justified update *or just* DJU *semantics. That is, a total interpretation M is a model of a MDLP $\mathcal{P}$ w.r.t. the DJU semantics whenever $M = least(Res(\mathcal{P}, M) \cup Def^\star(\mathcal{P}, M))$, where $Res(\mathcal{P}, M) = \uplus_{\mathcal{P}} \setminus Rej(\mathcal{P}, M)$ is the residue.*

The DJU semantics is the very first rule-rejecting semantics that has been used in dynamic logic programming. If we restrict to DLPs build from generalized logic programs, it is identical with the $\mathcal{P}$-justified updates semantics of [2]. Soon the original default assumptions showed to be too broad. In [4, 1], they have been replaced by $Def(\mathcal{P}, M)$. The semantics is formally defined as follows.

Definition 3. *A rule-rejecting semantics that uses $Rej(\mathcal{P}, M)$ for rejection and $Def(\mathcal{P}, M)$ for defaults is called* dynamic stable model *or just* DSM *semantics. Or equivalently, a total interpretation M is a model of a MDLP $\mathcal{P}$ w.r.t. the DSM semantics whenever $M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$, where the residue is as in Definition 2.*

In [5], the alternative notion of rejection, $Rej^\star(\mathcal{P}, M)$, has been combined with $Def^\star(\mathcal{P}, M)$ to produce semantics for DLPs build from extended logic programs. The semantics has been originally called update answer set semantics. In our setting we formalize it in Definition 4.

Definition 4. *A rule-rejecting semantics that uses $Rej^\star(\mathcal{P}, M)$ for rejection and $Def^\star(\mathcal{P}, M)$ for defaults is called* backward dynamic justified update *or just* BDJU *semantics. In other words, a total interpretation M is a model of a MDLP $\mathcal{P}$ w.r.t. the BDJU semantics whenever $M = least(Res^\star(\mathcal{P}, M) \cup Def^\star(\mathcal{P}, M))$, where $Res^\star(\mathcal{P}, M) = \Cup_{\mathcal{P}} \setminus Rej^\star(\mathcal{P}, M)$ is the residue.*

By the label "backward" we indicate use of $Rej^\star(\mathcal{P}, M)$ rejection, as the algorithm for its computation from [5] traverses $\mathcal{P}$ in backward direction compared to the one for $Rej(\mathcal{P}, M)$ found in [4, 1]. In [3], the three above mentioned semantics have been brought to a more general platform offered by GELPs. Also a backward variant of the DSM semantics has been introduced, that we formalize in Definition 5. In [3], this semantics is called U-model semantics.

Definition 5. *A rule-rejecting semantics that uses $Rej^\star(\mathcal{P}, M)$ for rejection and $Def(\mathcal{P}, M)$ for defaults is called* backward dynamic stable model *or just* BDSM *semantics. That is, a total interpretation M is a model of a MDLP $\mathcal{P}$ w.r.t. the BDSM semantics whenever $M = least(Res^\star(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$, where the residue is as in Definition 4.*

The set of all models of a program $\mathcal{P}$ w.r.t. the DJU semantics is denoted by $DJU(\mathcal{P})$. Similarly, $DSM(\mathcal{P})$, $BDJU(\mathcal{P})$ and $BDSM(\mathcal{P})$ are the sets of all models according to the remaining three semantics.

We have presented four rule-rejecting semantics of MDLPs. The following two examples taken from [3] show that each of this semantics is different.

Example 1. Let $\mathcal{P} = \{P_1 \prec P_2\}$ where $P_1 = \{a \leftarrow\}$, $P_2 = \{not\ a \leftarrow not\ a\}$. It holds that $DSM(\mathcal{P}) = BDSM(\mathcal{P}) = \{\{a, not \neg a\}\}$. But, for the other two, $DJU(\mathcal{P}) = BDJU(\mathcal{P}) = \{\{a, not \neg a\}, \{not\ a, not \neg a\}\}$.

Example 2. Let $\mathcal{P} = \{P_1 \prec P_2 \prec P_3\}$ where $P_1 = \{a \leftarrow\}$, $P_2 = \{not\ a \leftarrow\}$ and $P_3 = \{a \leftarrow a\}$. It holds that $DJU(\mathcal{P}) = DSM(\mathcal{P}) = \{\{not\ a, not \neg a\}\}$. On the other hand, $BDJU(\mathcal{P}) = BDSM(\mathcal{P}) = \{\{a, not \neg a\}, \{not\ a, not \neg a\}\}$.

Moreover, as it has been shown in [3], the sets of models assigned to arbitrary program $\mathcal{P}$, one set by each of these semantics, form a kind of hierarchy w.r.t. the set inclusion relation. The DSM semantics is the most restrictive one, the set of models w.r.t. DSM is always a subset of the other model-sets. On the other hand, the set of models w.r.t. any semantics is always a subset of the one w.r.t. BDJU, which always provides the broadest set of models. We summarize these observations in Theorem 1 taken from [3].

Theorem 1. *For each MDLP $\mathcal{P}$ it holds that*

$$DSM(\mathcal{P}) \subseteq DJU(\mathcal{P}) \subseteq BDJU(\mathcal{P}) \ ,$$
$$DSM(\mathcal{P}) \subseteq BDSM(\mathcal{P}) \subseteq BDJU(\mathcal{P}) \ .$$

4 Equality on the Class of Acyclic Programs

We have shown in Examples 1 and 2 that the four rule-rejecting semantics are in general distinct. However, many MDLPs exist, such as the one from Example 3, on which these four semantics coincide.

Example 3. Let $\mathcal{P} = \{P_1, P_2, P_3 \mid P_1 \prec P_3, P_2 \prec P_3\}$. Let $P_1 = \{a \leftarrow\}$, $P_2 = \{not\, a \leftarrow\}$ and $P_3 = \{a \leftarrow\}$. This simple MDLP can be viewed as a model of a community of three agents, who take part in the hierarchy of authorities. The first two of them are of incomparable authority and moreover, they have conflicting knowledge. This conflict is resolved by the third of them, who is represented by logic program P_3 and its authority level is superior to the former two. All four of the semantics agree with this intuition and assign $M = \{a, not\, \neg a\}$ to $\mathcal{P}$ as its single model.

Examples like this one lead us to a hypothesis that there probably are vast classes of programs on which several semantics coincide. It shows that several rule-rejecting semantics possibly behave equally on "plain" programs, that are not obfuscated with cyclic dependencies among literals or other obstacles. Different behavior on such programs is supposed to be caused by different ability of the semantics to deal with such obstacles.

To evaluate cyclic dependencies among literals in programs we adopt the graph-theoretic framework introduced in [5].

AND/OR-graph (N, C) is a hypergraph, whose set of nodes $N = N_A \uplus N_O$ decomposes into the set of *AND-nodes* N_A and the set of *OR-nodes* N_O, and its set of connectors $C = N \times \bigcup_{i=0}^{|N|} N^i$ is a function, i.e., for each $I \in N$ there is exactly one tuple $\langle O_1, \ldots, O_k\rangle$ s.t. $\langle I, O_1, \ldots, O_k\rangle \in C$. For any connector $\langle I, O_1, \ldots, O_k\rangle$, I is its *input node* and $O_1, \ldots, O_k$ are its *output nodes*.

Let (N, C) be an AND/OR-graph, $I \in N$ and $\langle I, O_1, \ldots, O_k\rangle \in C$. A tree p is a *path* in (N, C) *rooted in* I if one of the following conditions holds:

(i) $k = 0 \wedge p = \langle I\rangle$,
(ii) $k > 0 \wedge I \in N_A \wedge p = \langle I, p_1, \ldots, p_k\rangle$,
(iii) $k > 0 \wedge I \in N_O \wedge (\exists i)\ 1 \leq i \leq k \wedge p = \langle I, p_i\rangle$,

where p_i is a path in (N, C) rooted in O_i, $1 \leq i \leq k$.

Let $p = \langle I, p_1, \ldots, p_k\rangle$ be a path in an AND/OR-graph. Path p' is a *subpath* of p if $p' = p$ or p' is a subpath of p_i for some i, $1 \leq i \leq k$.

A path p in an AND/OR-graph is said to be *acyclic* if for every subpath p' (including p) rooted in the node R, no subpath p'' of p' is rooted in R.

Definition 6. *Let P be a logic program. An AND/OR-graph $G_P = (N, C)$ is* associated *with P if both of the following conditions hold:*

(i) $N_A = P \wedge N_O = \mathcal{L}$,
(ii) $C = \{\langle r, L_1, \ldots, L_k\rangle \mid r = L \leftarrow L_1, \ldots, L_k \in P\}$
$\cup\ \{\langle L, r_1, \ldots, r_n\rangle \mid \{r_1, \ldots, r_n\} = \{r \in P \mid h(r) = L\}\}$.

Armed with such a framework we instantly identify the class of acyclic programs in Definition 7. Clearly, this definition is equivalent to the original one, as introduced in [18].

Definition 7. *We say that logic program P is* strictly acyclic *(or just* acyclic*) if G_P does not contain a path that is cyclic. We say that a MDLP $\mathcal{P}$ is strictly acyclic if $\uplus_{\mathcal{P}}$ is strictly acyclic.*

In [5], further reduction of $G_{\mathcal{P}}$ is utilized, once an interpretation M and a given notion of rejection are available. The resulting reduced AND/OR-graph is stripped from dependencies corresponding to rules that are rejected or that are not applicable.

Definition 8. *Let $\mathcal{P}$ be a MDLP, M a total interpretation and $Rejected(\mathcal{P}, M)$ a set of rejected rules according to some rule-rejecting semantics. The* reduced AND/OR-graph *of $\mathcal{P}$ with respect to M, $G^M_{\mathcal{P}}$ is obtained from $G_{\mathcal{P}}$ by*

1. *removing all $r \in N_A$ and their connectors (as well as removing r from all connectors containing it as an output node) if either $r \in Rejected(\mathcal{P}, M)$ or $M \not\models b(r)$, and*
2. *replacing, for every $L \in \mathcal{O}$, the connector of not L by the 0-connector $\langle not\ L \rangle$, if L is associated with 0-connector after step 1 and no $r \in Rejected(\mathcal{P}, M)$ exists s.t. $h(r) = L$.*

Possessing the outlined framework, authors of [5] have introduced the "root condition" and the "chain condition", that we adopt in Definition 9 and 10 respectively.

Definition 9. *Let $\mathcal{P}$ be a MDLP, M a total interpretation and $Rejected(\mathcal{P}, M)$ a set of rejected rules according to some rule-rejecting semantics. We say that $\mathcal{P}$, M and $Rejected(\mathcal{P}, M)$ obey the* root condition *if, for each not $L \in M^-$, one of the following conditions holds:*

(i) $(\forall r \in \uplus_{\mathcal{P}})\ h(r) = L \implies M \not\models b(r)$,
(ii) there exists an acyclic path p in $G^M_{\mathcal{P}}$ rooted in not L.

Definition 10. *We say that a MDLP $\mathcal{P}$ and a total interpretation M obey the* chain condition *if, for each pair of rules $r \in P_i$, $r' \in P_j$ s.t. $i \prec j$, $r \bowtie r'$, $M \models b(r)$, $M \models b(r')$ and $r' \in Rej^\star(\mathcal{P}, M)$, there also exists $r'' \in P_s$ s.t. $j \prec s$, $r' \bowtie r''$ and $b(r'') \subseteq b(r)$.*

A theorem follows in [5], stating that if both root and chain condition are satisfied by a DLP $\mathcal{P}$, total interpretation M and $Rej(\mathcal{P}, M)$ then $M \in DSM(\mathcal{P})$ if and only if M is a model of $\mathcal{P}$ (both transformed to extended logic programs) w.r.t. the BDJU semantics.

In [3], relations between all four of these semantics are further investigated, once all four risen to the platform of GELPs. It is shown there, that the root condition renders a proper subclass of DLPs, in order to compare two semantics that utilize $Def(\mathcal{P}, M)$ and $Def^\star(\mathcal{P}, M)$ for defaults respectively, and share the

same implementation of rejection. We adopt this proposition from [3] and rise it to the platform of MDLPs in Theorem 2. In [3], it is also shown that two pairs of semantics that differ in rejection but use the same defaults, pairwise, coincide on a DLP $\mathcal{P}$ and a total interpretation M if they obey the chain condition. We adopt this proposition in Theorem 3.[2]

Theorem 2. *Let $\mathcal{P}$ be a MDLP, M a total interpretation. Then it holds that:*

(i) $M \in DJU(\mathcal{P}) \equiv M \in DSM(\mathcal{P})$ *if and only if* $\mathcal{P}$, M *and* $Rej(\mathcal{P}, M)$ *obey the root condition,*
(ii) $M \in BDJU(\mathcal{P}) \equiv M \in BDSM(\mathcal{P})$ *if and only if* $\mathcal{P}$, M *and* $Rej^{\star}(\mathcal{P}, M)$ *obey the root condition.*

Theorem 3. *Let $\mathcal{P}$ be a MDLP, M a total interpretation. If $\mathcal{P}$ and M obey the chain condition then each of the following propositions holds:*

(i) $M \in DJU(\mathcal{P}) \equiv M \in BDJU(\mathcal{P})$,
(ii) $M \in DSM(\mathcal{P}) \equiv M \in BDSM(\mathcal{P})$.

It follows in [3], that if both of the conditions are obeyed by $\mathcal{P}$ and M, then all four of the semantics coincide on $\mathcal{P}$ and M. However, as we show in Example 4, many times the chain condition is not obeyed but the semantics do coincide. We argue that this restriction is not accurate.

Example 4. Let $\mathcal{P} = \{P_1 \prec P_2 \prec P_3\}$, $P_1 = \{a \leftarrow\}$, $P_2 = \{not\, a \leftarrow\}$ and $P_3 = \{a \leftarrow not\, b\}$. The chain condition is not obeyed by $\mathcal{P}$ and $M = \{a, not\, b, not \neg a, not \neg b\}$. Yet, $DSM(\mathcal{P}) = BDSM(\mathcal{P}) = \{M\}$ and $DJU(\mathcal{P}) = BDJU(\mathcal{P}) = \{M\}$.

We now return to considerations about programs with restricted occurrence of cycles. We focus on a hypothesis that different behavior of semantics is always accompanied by presence of cyclic dependencies among literals. Our aim is to restrict somehow the occurrence of cyclic dependencies in order to establish the coincidence of the semantics.

Programs with cycles are often considered odd. Self-dependence, connected with presence of cycles, is marked as unpleasant and undesirable feature, as strict, deductive reasoning – closely interconnected with mathematical logic – forbids it. Yet, in logic programming cycles are useful for example to express equivalence. Moreover there are programs that contain cycles and still different semantics match regarding them. Both of these features are apparent from Example 5. Hence we introduce yet another, weaker, condition of acyclicity in the consecutive Definition 11. With this condition, we are able to identify programs, where cycles may be present, but each literal is supported by at least one acyclic derivation.

[2] We remark that this property does not depend on the particular choice of defaults. In fact, it holds for arbitrary set of default assumptions. See [10] for details.

Example 5. Let $\mathcal{P} = \{P_1 \prec P_2\}$, $P_1 = \{a \leftarrow b; b \leftarrow a\}$ and $P_2 = \{a \leftarrow\}$. All of the four semantics match on $\mathcal{P}$. $DJU(\mathcal{P}) = DSM(\mathcal{P}) = BDJU(\mathcal{P}) = BDSM(\mathcal{P}) = \{\{a, b, not\ \neg\ a, not\ \neg\ b\}\}$. Actually, the cyclic information of program P_1 is not redundant in any way. P_1 states that the truth value of a is equivalent with the truth value of b and vice versa. Later, when the more recent knowledge of P_2 appears telling that a is true we derive that also b is true.

Definition 11. *We say that logic program P is* sufficiently acyclic *if for every literal $L \in \mathcal{L}$ there exists an acyclic path in the hypergraph G_P associated with P that is rooted in L. A MDLP $\mathcal{P}$ is sufficiently acyclic whenever $\uplus_{\mathcal{P}}$ is sufficiently acyclic.*

Application of the condition of sufficient acyclicity on MDLPs in general is, however, useless – as when the residue is computed, several rules are retracted and the condition may not be satisfied any more. So we resort to the one-model relations of two semantics quite like in the case of the root condition. The relation is established for a program and a given model. Possessing a candidate-model, the residue is determined, and the condition is applied on the residue instead of the whole program.

To establish one-model equivalence of two semantics on a program, we repeatedly use a method, that is sketched in Remark 1.

Remark 1. Let $\mathcal{P}$ be a MDLP and let M be a total interpretation. Let S_1 and S_2 be two rule-rejecting semantics with shared implementation of defaults and different implementation of rejection. Let D be the set of defaults assigned to $\mathcal{P}$ and M by these semantics and let R_1 and R_2 be the residues assigned to $\mathcal{P}$ and M by S_1 and S_2 respectively. If

(i) $M \in S_2(\mathcal{P})$,
(ii) $R_1 \subseteq R_2$,

then $M \in S_1(\mathcal{P})$ if and only if there exists such $R \subseteq R_1$ that $M = least(R \cup D)$ – i.e., we are able to find R, a subset of R_1, s.t. R still contains enough of rules that are necessary to compute M. Therefore we concentrate on searching for such sets $R \subseteq R_1$ in order to establish equivalence of S_1 and S_2 regarding $\mathcal{P}$ and M.

The condition for one-model equality of that pairs of semantics that differ in the implementation of rejection and use same defaults is expressed in Theorem 4. The theorem uses the following lemma.

Lemma 1. *Let S be the BDSM or the BDJU semantics. Let $\mathcal{P}$ be a MDLP, $M \in S(\mathcal{P})$ and let $Defaults(\mathcal{P}, M)$ be default assumptions assigned to $\mathcal{P}$ and M by S. If the set R defined as*

$$R = \{r \mid r \in Res(\mathcal{P}, M) \wedge M \vDash b(r)\}$$

is sufficiently acyclic then M can be computed as a model in the given semantics using only the rules of R. That is, $M = least(R \cup Defaults(\mathcal{P}, M))$.

Proof. Since R is sufficiently acyclic, there exists a rule $r \in R$ such that for each $L \in b(r)$ for no $r' \in R$ holds $h(r') = L$. And $r \in R$ so it holds that $M \vDash b(r)$. From Definitions 2 and 4 and from how R is defined it follows that for each rule $q \in \mathit{Res}^{\star}(\mathcal{P}, M)$, $M \vDash b(q)$ there is a $q' \in R$ s.t. $h(q) = h(q')$ and since $M \in S(\mathcal{P})$ then $b(r) \subseteq \mathit{Defaults}(\mathcal{P}, M)$. We now construct

$$M^0 = \mathit{Defaults}(\mathcal{P}, M) \ , \qquad M^1 = M^0 \cup h(r) \ ,$$
$$R^0 = R \ , \qquad R^1 = R^0 \setminus \{r'' \mid h(r'') = h(r)\} \ .$$

Assume that M^j and R^j are constructed by adding one literal $L \in \mathcal{L}$ to M^{j-1} and removing all r'' from R^{j-1} such that $h(r'') = L$, $0 < j \leq i$. Again, as R is sufficiently acyclic, there is $r \in R^i$ s.t. for each $L \in b(r)$ for no $r' \in R^i$ holds $h(r') = L$. From the construction of D^i, R^i it follows that

$$(\forall j \leq i)\ M^j \cup \{h(r) \mid r \in R^j\} = M \ .$$

Therefore $b(r) \subseteq M^i$, and so we are able to construct

$$M^{i+1} = M^i \cup h(r) \ , \qquad R^{i+1} = R^i \setminus \{r'' \in R^i \mid h(r'') = h(r)\} \ .$$

It is straightforward that $\bigcup_{i=1}^{\infty} M^i = M$. This way we have computed M as a model in S only from the rules of R. (Step by step, we have simulated the iterations of the $\mathit{least}(\cdot)$ operator.) In other words,

$$M = \mathit{least}(R \cup \mathit{Defaults}(\mathcal{P}, M)) \ .$$

□

Theorem 4. *Let $\mathcal{P}$ be a MDLP and M be its total interpretation. If the set*

$$R = \{r \mid r \in \mathit{Res}(\mathcal{P}, M) \wedge M \vDash b(r)\}$$

is sufficiently acyclic then it holds that

(i) $M \in \mathit{DSM}(\mathcal{P}) \equiv M \in \mathit{BDSM}(\mathcal{P})$*, and also*
(ii) $M \in \mathit{DJU}(\mathcal{P}) \equiv M \in \mathit{BDJU}(\mathcal{P})$*.*

Proof. The only-if part of both (i) and (ii) follows from Theorem 1. The if part proves as follows. Let $\mathcal{P}$ be a MDLP. Let $M \in \mathit{BDSM}(\mathcal{P})$ ($\mathit{BDJU}(\mathcal{P})$ respectively). Let R be sufficiently acyclic. From Lemma 1 we get that M can be computed only using the rules of R. Since

$$R \subseteq \mathit{Res}(\mathcal{P}, M) \subseteq \mathit{Res}^{\star}(\mathcal{P}, M) \ ,$$

it follows from Remark 1 that $M \in \mathit{DSM}(\mathcal{P})$ ($M \in \mathit{DJU}(\mathcal{P})$). □

In Theorem 4 we have presented a restrictive condition for one-model equality of those pairs of semantics that differ in rejection and use same defaults. We now show (in Lemma 2) that under this condition also the root condition is satisfied. It follows as a direct consequence of this lemma and Theorem 4 that under our condition all four semantics coincide (Corollary 1).

Lemma 2. *Let $\mathcal{P}$ be a MDLP and M its total interpretation. Let*

$$R = \{r \mid r \in Res(\mathcal{P}, M) \wedge M \vDash b(r)\} \ .$$

If R is sufficiently acyclic then both of the triples $\mathcal{P}$, M, $Rej(\mathcal{P}, M)$ and $\mathcal{P}$, M, $Rej^{\star}(\mathcal{P}, M)$ obey the root condition.

Proof. R is sufficiently acyclic, hence for every $L \in M^{-}$ either $L \in Def(\mathcal{P}, M)$ and then condition (i) of Definition 9 (root condition) is satisfied or there exists a rule $r \in Res(\mathcal{P}, M)$ s.t. $M \vDash b(r)$ and $h(r) = L$ and therefore also $r' \in R$ s.t. $h(r') = L$ and so there is a path p in G_R rooted in L that is acyclic. The subpath p' of p, terminated in every *not* $L' \in \mathcal{D}$ whose connector was replaced by $\langle not\ L' \rangle$ in step 2 of the construction of $G^M_{\mathcal{P}}$, is an acyclic path in $G^M_{\mathcal{P}}$ rooted in L. And so condition (ii) of Definition 9 is satisfied. Hence the root condition is obeyed by $\mathcal{P}$, M and $Rej(\mathcal{P}, M)$.

As for each $r \in Res(\mathcal{P}, M)$, $M \vDash b(r)$ there exists such $r' \in Res^{\star}(\mathcal{P}, M)$ that $h(r') = h(r)$ and $M \vDash b(r')$ and vice versa, we get that also $\mathcal{P}$, M and $Rej^{\star}(\mathcal{P}, M)$ obey the root condition. □

Corollary 1. *Let $\mathcal{P}$ be a MDLP and M its total interpretation. If the set*

$$R = \{r \mid r \in Res(\mathcal{P}, M) \wedge M \vDash b(r)\}$$

is sufficiently acyclic then

$$M \in DSM(\mathcal{P}) \equiv M \in BDSM(\mathcal{P}) \equiv M \in DJU(\mathcal{P}) \equiv M \in BDJU(\mathcal{P}) \ .$$

Moreover, as for a strictly acyclic program each of its subsets is sufficiently acyclic, it trivially follows that all four semantics coincide on strictly acyclic programs as we state in the following corollary.

Corollary 2. *Let $\mathcal{P}$ be a strictly acyclic MDLP. Then*

$$DSM(\mathcal{P}) = BDSM(\mathcal{P}) = DJU(\mathcal{P}) = BDJU(\mathcal{P}) \ .$$

We have shown that the four rule-rejecting semantics coincide on strictly acyclic programs. In Corollary 1 we have also established a more accurate restriction that renders the one-model equivalence of the semantics. However, comparing entire model-sets assigned to a program by two semantics one by one is computationally as complex as computing and enumerating these two model-sets. So, this result is rather of theoretical value.

5 RDSM Semantics and DLPs

In [7], Alferes et al. have introduced a new semantics for linear DLPs. Motivation for this new semantics roots in the observation that even the most restrictive semantics, DSM, provides counterintuitive models for some programs (cf. Example 6).

Example 6. Let $\mathcal{P} = \{P_1 \prec P_2\}$ where $P_1 = \{a \leftarrow ; not\ a \leftarrow\}$ and $P_2 = \{a \leftarrow a\}$. It holds that $DSM(\{P_1\}) = \emptyset$, it is not surprising as P_1 is contradictory. If we inspect the single rule of P_2 we see that it actually brings no new factual information. We suppose that addition of such rule should not add new models to the program. However, $DSM(\mathcal{P}) = \{\{a, not\ \neg a\}\}$.

Such rules as the one of P_2 from Example 6, having head a subset of the body, are called *tautological*. Tautological rules are in fact just a special case of cycles that only span throughout one rule. In [7], authors have identified even broader class of extensions of DLPs that, according to their intuition, should not yield new models of the programs. Such extensions are called *refined extensions*. Then a principle has been formed, stating that, having a proper semantics, if a program $\mathcal{P}'$ is just a refined extension of $\mathcal{P}$ then it should not have a model that is not also a model of $\mathcal{P}$. This principle is called *refined extension principle*. We refer the reader who is interested in precise definitions to [7].

In [7], also a modified DSM semantics has been introduced. The modification is slight, two conflicting rules of the same program are allowed to reject each other. Formally, the set of rejected rules of this semantics is

$$Rej^R(\mathcal{P}, M) = \{r \in \dot{P}_i \mid (\exists r' \in \dot{P}_j)\ i \preceq j, M \vDash b(r'), r \bowtie r'\}\ .$$

The semantics is formalized in Definition 12.

Definition 12. *A rule-rejecting semantics of DLPs that uses $Rej^R(\mathcal{P}, M)$ for rejection and $Def(\mathcal{P}, M)$ for defaults is called* refined dynamic stable model *or just* RDSM *semantics. In other words, a total interpretation M is a model of a DLP $\mathcal{P}$ w.r.t. the RDSM semantics whenever $M = least(Res^R(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$, where $Res^R(\mathcal{P}, M)$ is the residue.*

We agree with [7] that the RDSM semantics is very favourable. It has been shown in [7] that it satisfies the refined extension principle and, as we adopt in Theorem 5, it always yields such model-set that is a subset of the model-set w.r.t. the DSM semantics. Moreover, it has been precisely described and motivated in [7], why some models provided by DSM should be excluded.

Theorem 5. *For any DLP $\mathcal{P}$ it holds that $RDSM(\mathcal{P}) \subseteq DSM(\mathcal{P})$.*

In [7], it further has been shown that for a program $\mathcal{P}$ that does not contain a pair of conflicting rules in the very same $P_i \in \mathcal{P}$, the RDSM and the DSM semantics coincide. However, this result neither is tight as many programs exist s.t. DSM and RDSM coincide on them and the condition is not satisfied.

The RDSM semantics has been introduced only for linear DLPs and according to our deepest knowledge all attempts to generalize it for MDLPs have failed so far (cf. [19]). Hence, in this section, we restrict our considerations to linear DLPs. In the remaining we show that under a very similar restriction as the one of Corollary 1, for a given model, all five of the semantics coincide.

First of all, the following example demonstrates why the condition has to be altered.

Example 7. Recall again the program $\mathcal{P}$ from Example 6. Let $M = \{a, not\ \neg a\}$. Even if $R = \{a \leftarrow, a \leftarrow a\}$ is sufficiently acyclic, $M \in DSM(P)$ and $M \notin RDSM(P)$. Indeed, the fact that $R \not\subseteq Res^R(\mathcal{P}, M)$ causes the trouble. The sufficient acyclicity is broken in $Res^R(\mathcal{P}, M)$ and therefore a can not be derived in the refined semantics.

The further restrictive condition is introduced in Theorem 6, where we prove the one-model coincidence of RDSM and DSM and we also confirm that the propositions of Theorem 4 hold under this modified condition as well. The theorem uses the following lemma.

Lemma 3. *Let semantics S be one of DSM, DJU, BDSM and BDJU. Let $\mathcal{P}$ be a DLP. Let $M \in S(\mathcal{P})$. Let $Rejected(\mathcal{P}, M)$ be rejected rules, $Residue(\mathcal{P}, M)$ be the residue and $Defaults(\mathcal{P}, M)$ be defaults assigned to $\mathcal{P}$ and M by S. If*

$$R' = \{r \mid r \in Res^R(\mathcal{P}, M) \wedge M \vDash b(r)\}$$

is sufficiently acyclic then M can be computed as a model in the given semantics using only the rules of R'. That is, $M = least(R' \cup Defaults(\mathcal{P}, M))$.

Proof. From Definitions 2, 4 and 12 and from how R' is defined it follows that if $M \in S(\mathcal{P})$ then for each rule $q \in Residue(\mathcal{P}, M)$, $M \vDash b(q)$ there is a $q' \in R'$ s.t. $h(q) = h(q')$. Once we are aware of this fact this lemma is proved exactly as Lemma 1. □

Theorem 6. *Let $\mathcal{P}$ be a DLP and M be its total interpretation. If*

$$R' = \{r \mid r \in Res^R(\mathcal{P}, M) \wedge M \vDash b(r)\}$$

is sufficiently acyclic then the following propositions hold:

(i) $M \in DSM(\mathcal{P}) \equiv M \in RDSM(\mathcal{P})$,
(ii) $M \in DSM(\mathcal{P}) \equiv M \in BDSM(\mathcal{P})$,
(iii) $M \in DJU(\mathcal{P}) \equiv M \in BDJU(\mathcal{P})$.

Proof. Propositions (ii) and (iii) are proved like in the above Theorem 4. The if part of (i) follows from Theorem 5. The only if part of (i) proves as follows.

Let $M \in DSM(\mathcal{P})$. Let R' be sufficiently acyclic. From Lemma 3 we know that M can be computed using only the rules of R'. Also

$$R' \subseteq Res^R(\mathcal{P}, M) \subseteq Res(\mathcal{P}, M) \ ,$$

so it follows from Remark 1 that $M \in RDSM(\mathcal{P})$. □

In the following lemma we show that even if we have slightly modified the condition, its satisfaction still implies that the root condition is also satisfied. Hence if the condition is satisfied, all five of the semantics for DLPs coincide on a given model as we state in Corollary 3.

Lemma 4. *Let $\mathcal{P}$ be a MDLP and M its total interpretation. Let*

$$R' = \{r \mid r \in Res^R(\mathcal{P}, M) \wedge M \vDash b(r)\} \ .$$

If R' is sufficiently acyclic and $M \in DJU(\mathcal{P})$ ($M \in BDJU(\mathcal{P})$) then $\mathcal{P}$, M, $Rej(\mathcal{P}, M)$ ($\mathcal{P}$, M, $Rej^\star(\mathcal{P}, M)$) obey the root condition.

Proof. This lemma the same way as Lemma 2 if we realize that when $M \in DJU(\mathcal{P})$ ($M \in BDJU(\mathcal{P})$) then for each rule $r \in Res(\mathcal{P}, M)$ ($r \in Res^\star(\mathcal{P}, M)$) s.t. $M \vDash b(r)$ and $h(r) = L$ there also exists $r' \in R'$ s.t. $h(r') = L$. □

Corollary 3. *Let $\mathcal{P}$ be a DLP and M its total interpretation. If the set*

$$R' = \{r \mid r \in Res^R(\mathcal{P}, M) \wedge M \vDash b(r)\}$$

is sufficiently acyclic then

$$M \in DSM(\mathcal{P}) \equiv M \in BDSM(\mathcal{P}) \equiv M \in RDSM(\mathcal{P}) \equiv$$
$$\equiv M \in DJU(\mathcal{P}) \equiv M \in BDJU(\mathcal{P}) \ .$$

As for Corollary 1, also for Corollary 3 it holds that if, using it, we want to compare entire model-sets assigned to a program by a pair of semantics, computational complexity is the same as enumerating and comparing these two model-sets. Anyway, it trivially follows from this corollary that all five of the semantics coincide on strictly acyclic programs, as follows in Corollary 4.

Corollary 4. *Let $\mathcal{P}$ be a strictly acyclic DLP. Then*

$$DSM(\mathcal{P}) = BDSM(\mathcal{P}) = RDSM(\mathcal{P}) = DJU(\mathcal{P}) = BDJU(\mathcal{P}) \ .$$

6 Conclusion

In accord with [3, 15], we have built MDLPs over a more general language of GELPs, that allows for more elegant comparisons, since no transformations are necessary as the previous approaches are obtained as its special cases. We have then compared four different rule-rejecting semantics of MDLPs and in addition one more when restricted to linear DLPs. We have introduced sufficient acyclicity. Using this notion, we have provided a restrictive condition on a MDLP (DLP) $\mathcal{P}$ and a given candidate model M s.t. if it is satisfied all four (five) semantics coincide on $\mathcal{P}$ and M. As a trivial consequence we have stated the main result, that on strictly acyclic programs all four (five) of the semantics coincide.

There are several open problems. As there are programs that contain cycles and several of the five semantics coincide on them, the search for proper characterization of the class of programs on which these semantics coincide is still open. In this line, we suggest investigation of other well known classes, as stratified and call-consistent programs. One of the most favourable semantics, RDSM, is only known for DLPs, generalizing RDSM to MDLPs is a challenging problem.

Comparing semantics that are based on rejection of rules with other approaches (such as the one of [15] based on Kripke structures) might be interesting. To meet this goal, we propose that more abstract criteria for evaluating these semantics should be introduced, seeing some of the present ones, e.g., the refined extension principle of [6, 7], too attached to the rule-rejecting framework.

Acknowledgements

I would like to thank to anonymous referees for valuable comments and suggestions. I would like to thank to Ján Šefránek and João A. Leite for their advising and help and to Michaela Danišová and to Martin Baláž for language and typographical corrections.

References

1. Leite, J.A., Alferes, J.J., Pereira, L.M.: Multi-dimensional dynamic logic programming. In Sadri, F., Satoh, K., eds.: Proceedings of the CL-2000 Workshop on Computational Logic in Multi-Agent Systems (CLIMA'00). (2000) 17–26 http://centria.di.fct.unl.pt/~jleite/papers/clima00.ps.gz.
2. Leite, J.A., Pereira, L.M.: Iterated logic program updates. In Jaffar, J., ed.: Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming (JICSLP'98), MIT Press (1998) 265–278 http://centria.di.fct.unl.pt/~jleite/papers/jicslp98.ps.gz.
3. Leite, J.A.: Evolving Knowledge Bases: Specification and Semantics. Volume 81 of Frontiers in Artificial Intelligence and Applications, Dissertations in Artificial Intelligence. IOS Press, Amsterdam (2003)
4. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic logic programming. In Cohn, A.G., Schubert, L.K., Shapiro, S.C., eds.: Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Morgan Kaufmann (1998) 98–109 http://centria.di.fct.unl.pt/~jleite/papers/kr98.ps.gz.
5. Eiter, T., Sabbatini, G., Fink, M., Tompits, H.: On updates of logic programs: Semantics and properties. Technical Report 1843-00-08, Institute of Information Systems, Vienna University of Technology (2002) http://www.kr.tuwien.ac.at/research/reports/rr0008.ps.gz.
6. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Studia Logica **1** (2005) http://centria.di.fct.unl.pt/~jleite/papers/sl05.ps.gz.
7. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: Semantics for dynamic logic programming: A principle-based approach. In Lifschitz, V., Niemela, I., eds.: Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7), Springer-Verlag (2004) http://centria.di.fct.unl.pt/~jleite/papers/lpnmr7.ps.gz.
8. Leite, J.: On some differences between semantics of logic program updates. In Lemaitre, C., Reyes, C.A., Gonzalez, J.A., eds.: Advances in Artificial Intelligence: Proceedings of the 9th Ibero-American Conference on AI (IBERAMIA-04). LNAI, Springer (2004) http://centria.di.fct.unl.pt/~jleite/papers/iberamia04.ps.gz.

9. Eiter, T., Sabbatini, G., Fink, M., Tompits, H.: On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming (2002) 711–767 http://arxiv.org/abs/cs/?0109006.
10. Homola, M.: On relations of the various semantic approaches in multidimensional dynamic logic programming. Master's thesis, Comenius University, Faculty of Mathematics Physics and Informatics, Bratislava (2004) http://tbc.sk/papers/masters_thesis.pdf.
11. Leite, J.A., Alferes, J.J., Pereira, L.M.: Multi-dimensional logic programming. Technical report, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2001) http://centria.di.fct.unl.pt/~jleite/papers/mdlp-tr.ps.gz.
12. Leite, J.A., Alferes, J.J., Pereira, L.M.: Multi-dimensional dynamic knowledge representation. In Eiter, T., Faber, W., Truszczynski, M., eds.: Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01), Springer (2001) 365–378 http://centria.di.fct.unl.pt/~jleite/papers/lpnmr01.ps.gz.
13. Alferes, J.J., Banti, F., Brogi, A.: From logic program updates to action description updates. In Leite, J.A., Torroni, P., eds.: CLIMA V Computational Logic in Multi-agent Systems. (2004) 243–259 http://centria.di.fct.unl.pt/~jleite/climaV/climaV-preprocs.pdf.
14. Sakama, C., Inoue, K.: Coordination between logical agents. In Leite, J.A., Torroni, P., eds.: CLIMA V Computational Logic in Multi-agent Systems. (2004) 98–113 http://www.sys.wakayama-u.ac.jp/~sakama/papers/clima04.ps.gz.
15. Šefránek, J.: Semantic considerations on rejection. In: Proceedings of the International Workshop on Non-Monotonic Reasoning (NMR 2004), Foundations of Nonmonotonic Reasoning. (2004) http://www.pims.math.ca/science/2004/NMR/papers/paper48.pdf.
16. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. Journal of the ACM **23** (1976) 733–742
17. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K.A., eds.: Logic Programming, Proceedings of the Fifth International Conference and Symposium, MIT Press (1988) 1070–1080 http://www.cs.ttu.edu/~mgelfond/papers/stable.ps.
18. Apt, K.R., Bezem, M.: Acyclic programs. New Generation Computing **9** (1991) 335–363
19. Šiška, J.: Refined extension principle for multi-dimensional dynamic logic programming. Master's thesis, Comenius University, Faculty of Mathematics Physics and Informatics, Bratislava (2004) http://people.ksp.sk/~yoyo/dipl/dipl.pdf.