Multi-dimensional Dynamic Knowledge Representation

João Alexandre Leite, José Júlio Alferes, and Luís Moniz Pereira

Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
{jleite,jja,lmp}@di.fct.unl.pt

Abstract. According to *Dynamic Logic Programming (DLP)*, knowledge may be given by a sequence of theories (encoded as logic programs) representing different states of knowledge. These may represent time (e.g. in updates), specificity (e.g. in taxonomies), strength of updating instance (e.g. in the legislative domain), hierarchical position of knowledge source (e.g. in organizations), etc. The mutual relationships extant among states are used to determine the semantics of the combined theory composed of all the individual theories. Although suitable to encode a single dimension (e.g. time, hierarchies...), *DLP* cannot deal with more than one simultaneously because it is defined only for a linear sequence of states. To overcome this limitation, we introduce the notion of *Multi-dimensional Dynamic Logic Programming ($\mathcal{MDLP}$)*, which generalizes *DLP* to collections of states organized in arbitrary acyclic digraphs representing precedence. In this setting, $\mathcal{MDLP}$ assigns semantics to sets and subsets of such logic programs. By dint of this natural generalization, $\mathcal{MDLP}$ affords extra expressiveness, in effect enlarging the latitude of logic programming applications unifiable under a single framework. The generality and flexibility provided by the acyclic digraphs ensures a wide scope and variety of application possibilities.

1 Introduction and Motivation

In [1], the paradigm of *Dynamic Logic Programming (DLP)* was introduced, following the eschewing of performing updates on a model basis, as in [8,15,16,19], but rather as a process of logic programming rule updates [13].

According to *Dynamic Logic Programming* (*DLP*), itself a generalization of the notion of the update of a logic program P by another one U, knowledge is given by a series of theories (encoded as generalized logic programs) representing distinct supervenient states of the world. Different states, sequentially ordered, can represent different time periods [1], different agents [9], different hierarchical instances [17], or even different domains of knowledge [12]. Consequently, individual theories may comprise mutually contradictory as well as overlapping information. The role of *DLP* is to employ the mutual relationships extant among different states to precisely determine the declarative as well as the procedural semantics for the combined theory comprised of all individual theories at each

state. Intuitively, one can add, at the end of the sequence, newer or more specific rules (arising from new, renewly acquired, or more specific knowledge) leaving to *DLP* the task of ensuring that these added rules are in force, and that previous or less specific rules are still valid (by inertia) only so far as possible, i.e. that they are kept for as long as they are not in conflict with newly added ones, these always prevailing. The common feature among the applications of *DLP* is that the states associated with the given set of theories encode only one of several possible representational dimensions (e.g. time, hierarchies, domains,...).

For example, *DLP* can be used to model the relationship of a group of agents related according to a linear hierarchy, and *DLP* can be used to model the evolution of a single agent over time. But *DLP*, as it stands, cannot deal with both settings at once, and model the evolution of one such group of agents over time, inasmuch *DLP* is defined for linear sequences of states alone. Nor can it model hierarchical relations amongst agents that have more than one superior (and multiple inheritance). An instance of a multi-dimensional scenario is legal reasoning, where legislative agency is divided conforming to a hierarchy of power, governed by the principle *Lex Superior (Lex Superior Derogat Legi Inferiori)* by which the rule issued by a higher hierarchical authority overrides the one issued by a lower one, and the evolution of law in time is governed by the principle *Lex Posterior (Lex Posterior Derogat Legi Priori)* by which the rule enacted at a later point in time overrides the earlier one. *DLP* can be used to model each of these principles separately, by using the sequence of states to represent either the hierarchy or time, but is unable to cope with both at once when they interact.

In effect, knowledge updating is not to be simply envisaged as taking place in the time dimension alone. Several updating dimensions may combine simultaneously, with or without the temporal one, such as specificity (as in taxonomies), strength of the updating instance (as in the legislative domain), hierarchical position of the knowledge source (as in organizations), credibility of the source (as in uncertain, mined, or learnt knowledge), or opinion precedence (as in a society of agents). For this combination to be possible, *DLP* needs to be extended to allow for a more general structuring of states.

In this paper we introduce the notion of *Multi-dimensional Dynamic Logic Programming ($\mathcal{MDLP}$)* which generalizes *DLP* to cater for collections of states represented by arbitrary directed acyclic graphs. In this setting, $\mathcal{MDLP}$ assigns semantics to sets and subsets of logic programs, depending on how they stand in relation to one another, this relation being defined by the acyclic digraph (DAG) that configures the states. By dint of such a natural generalization, $\mathcal{MDLP}$ affords extra expressiveness, thereby enlarging the latitude of logic programming applications unifiable under a single framework. The generality and flexibility provided by DAGs ensures a wide scope and variety of possibilities.

The remainder of this paper is structured as follows: in Section 2 we introduce some background definitions; in Section 3 we introduce $\mathcal{MDLP}$ and proffer a declarative semantics; in Section 4 some illustrative examples are presented; in Section 5 an equivalent semantics based on a syntactical transformation is provided, proven sound and complete wrt. the declarative semantics; in Section

6 we set forth some basic properties; in Section 7 we conclude and open the doors of future developments.

2   Background

Generalized Logic Programs and Their Stable Models To represent *negative* information in logic programs and in their updates, since we need to allow default negation *not A* not only in premises of their clauses but also in their heads, we use *generalized logic programs* as defined in [1][1].

By a *generalized logic program* P in a language $\mathcal{L}$ we mean a finite or infinite set of propositional clauses of the form $L_0 \leftarrow L_1, \ldots, L_n$ where each L_i is a literal (i.e. an atom A or the default negation of an atom *not A*). If r is a clause (or rule), by $H(r)$ we mean L, and by $B(r)$ we mean $L_1, \ldots, L_n$. If $H(r) = A$ (resp. $H(r) = not\,A$) then $not\,H(r) = not\,A$ (resp. $not\,H(r) = A$). By a (2-valued) *interpretation* M of $\mathcal{L}$ we mean any set of literals from $\mathcal{L}$ that satisfies the condition that for any A, *precisely one* of the literals A or $not\,A$ belongs to M. Given an interpretation M we define $M^+ = \{A : A$ is an atom, $A \in M\}$ and $M^- = \{not\,A : A$ is an atom, $not\,A \in M\}$. Following established tradition, wherever convenient we omit the default (negative) atoms when describing interpretations and models. We say that a (2-valued) interpretation M of $\mathcal{L}$ is a stable model of a generalized logic program P if $\rho(M) = least\,(\rho(P) \cup \rho(M^-))$, where $\rho(.)$ univocally renames every default literal *not A* in a program or model into new atoms, say not_A. The class of generalized logic programs can be viewed as a special case of yet broader classes of programs, introduced earlier in [7] and in [14], and, for the special case of normal programs, their semantics coincides with the stable models one [6].

Graphs A *directed graph*, or *digraph*, $D = (V, E)$ is a pair of two finite or infinite sets $V = V_D$ of *vertices* and $E = E_D$ of pairs of vertices or (*directed*) *edges*. A *directed edge sequence from v_0 to v_n* in a digraph is a sequence of edges $e_1, e_2, ..., e_n \in E_D$ such that $e_i = (v_{i-1}, v_i)$ for $i = 1, ..., n$. A *directed path* is a directed edge sequence in which all the edges are distinct. A *directed acyclic graph*, or *acyclic digraph (DAG)*, is a digraph D such that there are no directed edge sequences from v to v, for all vertices v of D. A *source* is a vertex with in-valency 0 (number of edges for which it is a final vertex) and a *sink* is a vertex with out-valency 0 (number of edges for which it is an initial vertex). We say that $v < w$ if there is a directed path from v to w and that $v \leq w$ if $v < w$ or $v = w$. The *transitive closure* of a graph D is a graph $D^+ = (V, E^+)$ such that for all $v, w \in V$ there is an edge (v, w) in E^+ if and only if $v < w$ in D. The relevancy DAG of a DAG D wrt a vertex v of D is $D_v = (V_v, E_v)$ where $V_v = \{v_i : v_i \in V$ and $v_i \leq v\}$ and $E_v = \{(v_i, v_j) : (v_i, v_j) \in E$ and $v_i, v_j \in V_v \}$. The relevancy DAG of a DAG D wrt a set of vertices S of D is $D_S = (V_S, E_S)$

---

[1] In [2] the reader can find the motivation for the usage of generalized logic programs, instead of using simple denials by freely moving the head *not*s into the body.

where $V_S = \bigcup_{v \in S} V_v$ and $E_S = \bigcup_{v \in S} E_v$, where $D_v = (V_v, E_v)$ is the relevancy DAG of D wrt v.

3   Multi-dimensional Dynamic Logic Programming

As noted in the introduction, allowing the individual theories of a dynamic program update to relate via a linear sequence of states only, delimits DLP to represent and reason about a single aspect of a system (e.g. time, hierarchy,...). In this section we generalize DLP to allow for states represented by the vertices of a DAG, and their precedence relations by the corresponding edges, thus enabling concurrent representation, depending on the choice of a particular DAG, of several dimensions of an updatable system. In particular, the DAG can stand not only for a system of n independent dimensions, but also for inter-dimensional precedence. In this setting, $\mathcal{MDLP}$ assigns semantics to sets and subsets of logic programs, depending on how they so relate to one another.

We start by defining the framework consisting of the generalized logic programs indexed by a DAG. Throughout this paper, we will restrict ourselves to DAGs such that, for every vertex v of the DAG, any path ending in v is finite.

Definition 1 (Multi-dimensional Dynamic Logic Program). *Let $\mathcal{L}$ be a propositional language. A* Multi-dimensional Dynamic Logic Program (MDLP), *$\mathcal{P}$, is a pair $(\mathcal{P}_D, D)$ where $D = (V, E)$ is a DAG and $\mathcal{P}_D = \{P_v : v \in V\}$ is a set of generalized logic programs in the language $\mathcal{L}$, indexed by the vertices $v \in V$ of D. We call* states *such vertices of D. For simplicity, we often leave $\mathcal{L}$ implicit.*

3.1   Declarative Semantics

To characterize the models of $\mathcal{P}$ at any given state we will keep to the basic intuition of logic program updates, whereby an interpretation is a stable model of the update of a program P by a program U iff it is a stable model of a program consisting of the rules of U together with a subset of the rules of P, comprised by all those that are not rejected due to their being overridden by program U i.e. that do not carry over by inertia. With the introduction of a DAG to index programs, a program may have more than a single ancestor. This has to be dealt with, the desired intuition being that a program $P_v \in \mathcal{P}_D$ can be used to reject rules of any program $P_u \in \mathcal{P}_D$ if there is a directed path from u to v. Moreover, if some atom is not defined in the update nor in any of its ancestor, its negation is assumed by default. Formally, the stable models of the *MDLP* are:

Definition 2 (Stable Models at state s). *Let $\mathcal{P} = (\mathcal{P}_D, D)$ be a MDLP, where $\mathcal{P}_D = \{P_v : v \in V\}$ and $D = (V, E)$. An interpretation M_s is a stable model of $\mathcal{P}$ at state $s \in V$, iff*

$M_s = least \left([\mathcal{P}_s - Reject(s, M_s)] \cup Default\,(\mathcal{P}_s, M_s) \right)$ where A is an atom and:

$\mathcal{P}_s = \bigcup_{i \leq s} P_i$

$Reject(s, M_s) = \{r \in P_i \mid \exists r' \in P_j, i < j \leq s, H(r) = not\, H(r') \wedge M_s \vDash B(r')\}$

$Default\,(\mathcal{P}_s, M_s) = \{not\, A \mid \nexists r \in \mathcal{P}_s : (H(r) = A) \wedge M_s \vDash B(r)\}$

Intuitively, the set $Reject(s, M_s)$ contains those rules belonging to a program indexed by a state i that are overridden by the head of another rule with true body in state j along a path to state s. $\mathcal{P}_s$ contains all rules of all programs that are indexed by a state along all paths to state s, i.e. all rules that are potentially relevant to determine the semantics at state s. The set $Default\,(\mathcal{P}_s, M_s)$ contains default negations $not\, A$ of all unsupported atoms A, i.e., those atoms A for which there is no rule in $\mathcal{P}_s$ whose body is true in M_s.

Example 1. Consider the diamond shaped *MDLP* $\mathcal{P} = (\mathcal{P}_D, D)$ such that $\mathcal{P}_D = \{P_t, P_u, P_v, P_w\}$ and $D = (\{t, u, v, w\}, \{(t, u), (t, v), (u, w), (v, w)\})$ where

$$P_t = \{d \leftarrow\} \quad P_u = \{a \leftarrow not\, e\} \quad P_v = \{not\, a \leftarrow d\}$$
$$P_w = \{not\, a \leftarrow b; b \leftarrow not\, c; c \leftarrow not\, b\}$$

The only stable model at state w is $M_w = \{not\, a, b, not\, c, d, not\, e\}$. In fact, we have that $Reject(w, M_w) = \{a \leftarrow not\, e\}$ and $Default\,(\mathcal{P}_w, M_w) = \{not\, c, not\, e\}$ and, finally,

$$[\mathcal{P}_w - Reject(s, M_w)] \cup Default\,(\mathcal{P}_w, M_w) =$$
$$= \{d \leftarrow; not\, a \leftarrow d; not\, a \leftarrow b; b \leftarrow not\, c; c \leftarrow not\, b\} \cup \{not\, c, not\, e\}$$

whose least model is M_w. M_w is *the only* stable model at state w.

The next proposition establishes that to determine the models of a *MDLP* at state s, we need only consider the part of the *MDLP* corresponding to the relevancy graph wrt state s.

Proposition 1. *Let* $\mathcal{P} = (\mathcal{P}_D, D)$ *be a* MDLP, *where* $\mathcal{P}_D = \{P_v : v \in V\}$ *and* $D = (V, E)$. *Let s be a state in V. Let* $\mathcal{P}' = (\mathcal{P}_{D_s}, D_s)$ *be a* MDLP *where* $D_s = (V_s, E_s)$ *is the relevancy DAG of D wrt s, and* $\mathcal{P}_{D_s} = \{P_v : v \in V_s\}$. *M is a stable model of* $\mathcal{P}$ *at state s iff M is a stable model of* $\mathcal{P}'$ *at state s.*

We might have a situation where we desire to determine the semantics jointly at more than one state. If all these states belong to the relevancy graph of one of them, we simply determine the models at that state (Prop. 1). But this might not be the case. Formally, the semantics of a *MDLP* at an arbitrary set of its states is determined by the definition:

Definition 3 (Stable Models at a set of states S). *Let* $\mathcal{P} = (\mathcal{P}_D, D)$ *be a* MDLP, *where* $\mathcal{P}_D = \{P_v : v \in V\}$ *and* $D = (V, E)$. *Let S be a set of states such*

that $S \subseteq V$. An interpretation M_S is a stable model *of $\mathcal{P}$ at the set of states S iff $M_S = least\left([\mathcal{P}_S - Reject(S, M_S)] \cup Default\,(\mathcal{P}_S, M_S)\right)$ where:*

$$\mathcal{P}_S = \bigcup_{s \in S} \left(\bigcup_{i \leq s} P_i \right)$$

$$Reject(S, M_S) = \left\{ \begin{matrix} r \in P_i \mid \exists s \in S, \exists r' \in P_j, i < j \leq s, \\ H(r) = not\,H(r') \wedge M_S \vDash B(r') \end{matrix} \right\}$$

$$Default\,(\mathcal{P}_S, M_S) = \{not\,A \mid \nexists r \in \mathcal{P}_S : (H(r) = A) \wedge M_S \vDash B(r)\}$$

This is equivalent to the addition of a new vertex α to the DAG, and connecting to α, by addition of edges, all states we wish to consider. Furthermore, the program indexed by α is empty. We then determine the stable models of this new $MDLP$ at state α. In Section 6, we provide semantics preserving simplifications of these definitions, according to which only a subset of these newly added edges is needed. Note the addition of state α does not affect the stable models at other states. Indeed, α and the newly introduced edges do not belong to the relevancy DAG wrt. any other state. A particular case of the above definition is when $S = V$, corresponding to the semantics of the whole $MDLP$.

4   Illustrative Examples

By its very motivation and design, $\mathcal{MDLP}$ is well suited for combining knowledge arising from various sources, specially when some of these sources have priority over the others. More precisely, when rules from some sources are used to reject rules of other, less prior, sources. In particular, $\mathcal{MDLP}$ is well suited for combining knowledge originating within hierarchically organized sources, as the following schematic example illustrates, which combines knowledge coming from diverse sectors of such an organization.

Example 2. Consider a company with a president, a board of directors and (at least) two departments: the quality management and financial ones.

To improve the quality of the products produced by the company, the quality management department has decided not to buy any product whose reliability is less than guaranteed. In other words, it has adopted the rule[2]:

$$not\,buy(X) \leftarrow not\,reliable(X)$$

On the other hand, to save money, the financial department has decided to buy products of a type in need if they are cheap, viz.

$$buy(X) \leftarrow type(X, T), needed(T), cheap(X)$$

The board of directors, in order to keep production going, has decided that whenever there is still a need for a type of product, exactly one product of that type must be bought. This can be coded by the following logic programming rules, stating that if X is a product of a needed type, and if the need for that

---

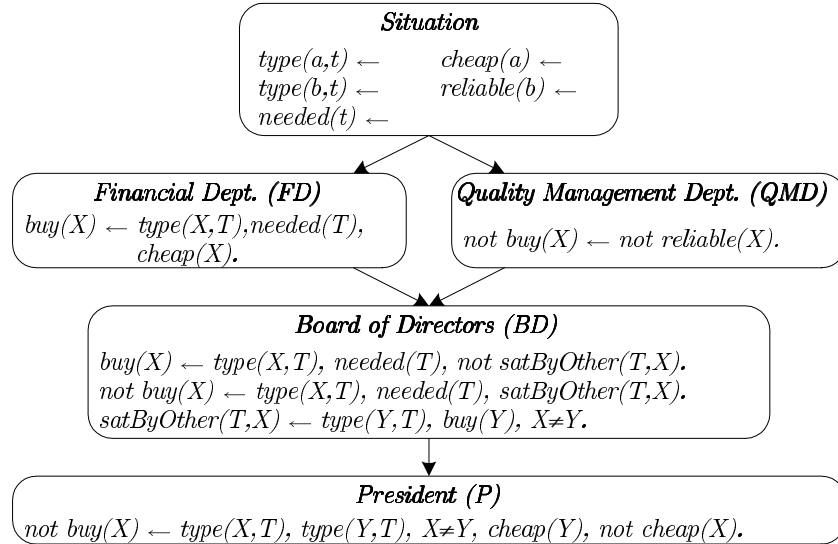[2] Rules with variables stand for the set of all their ground instances.

Fig. 1.

type of product has not been already satisfied by buying some other product of that type, then X must be bought; if the need is satisfied by buying some other product of that type, then X should not be bought:

$$buy(X) \leftarrow type(X,T), needed(T), not\, satByOther(T, X)$$
$$not\, buy(X) \leftarrow type(X,T), needed(T), satByOther(T, X)$$
$$satByOther(T, X) \leftarrow type(Y,T), buy(Y), X \neq Y$$

Finally, the president decided for the company never to buy products that have a cheap alternative. I.e. if two products are of the same type, and only one of them is cheap, the company should not buy the other:

$$not\, buy(X) \leftarrow type(X,T), type(Y,T), X \neq Y, cheap(Y), not\, cheap(X)$$

Suppose further that there are two products, a and b, the first being cheap and the latter reliable, both of type t and both of needed type t.

According to the company's organizational chart, the rules of the president can overrule those of all other sectors, and those established by the board can overrule those decided by the departments. No department has precedence over any other. This situation can easily be modeled by the MDLP of Figure 1.

To know what would be the decision of each of the sectors about which products to buy, not taking under consideration the deliberation of its superiors, all needs to be done is to determine the stable models at the state corresponding to that sector. For example, the reader can check that at state QMD there is a single stable model in which both $not\, buy(a)$ and $not\, buy(b)$ are true. At the

state BD there are two stable models: one in which $buy(a)$ and $not\,buy(b)$ are true; another where $not\,buy(a)$ and $buy(b)$ are true instead.

More interesting would be to know what is the decision of the company as a whole, when taking into account the rules of all sectors and their hierarchical organization. This is reflected by the stable models of the whole *MDLP*, i.e. the stable models at the set of all states of the *MDLP*. The reader can check that, in this instance, there is a single stable model in which $buy(a)$ and $not\,buy(b)$ are true. It coincides with the single stable model at state *president* because all other states belong to its relevancy graph. □

The next example describes how $\mathcal{MDLP}$ can deal with collision principles, e.g. found in legal reasoning, such as *Lex Superior (Lex Superior Derogat Legi Inferiori)* according to which the rule issued by a higher hierarchical authority overrides the one issued by a lower one, and *Lex Posterior (Lex Posterior Derogat Legi Priori)* according to which the rule enacted at a later point in time overrides the earlier one, i.e how the combination of a temporal and an hierarchical dimensions can be combined into a single *MDLP*.

Example 3. In February 97, the President of Brazil (PB) passed a law determining that, in order to guarantee the safety aboard public transportation airplanes, all weapons were forbidden. Furthermore, all exceptional situations that, due to public interest, require an armed law enforcement or military agent are to be the subject of specific regulation by the Military and Justice Ministries. We will refer to this as rule 1. At the time of this event, there was in force an internal norm of the Department of Civil Aviation (DCA) stating that "Armed Forces Officials and Police Officers can board with their weapons if their destination is a national airport". We will refer to this as rule 2. Restricting ourselves to the essential parts of these regulations, they can be encoded by the generalized logic program clauses:

$$rule1 : not\,carry\_weapon \leftarrow not\,exception$$
$$rule2 : carry\_weapon \leftarrow armed\_officer$$

Let us consider a lattice with two distinct dimensions, corresponding to the two principles governing this situation: *Lex Superior (d_1)* and *Lex Posterior (d_2)*. Besides the two agencies involved in this situation (PB and DCA), we will consider two time points representing the time when the two regulations were enacted. We have then a graph whose vertices are $\{(PB, 1), (PB, 2), (DCA, 1), (DCA, 2)\}$ (in the form (agency,time)) as portrayed in Fig.2. We have that $P_{DCA,1}$ contains rule 2, $P_{PB,2}$ contains rule 1 and the other two programs are empty. Let us further assume that there is an armed_officer represented by a fact in $P_{DCA,1}$. Applying Def.2, for $M_{PB,2} = \{not\,carry\_weapon,\ not\,exception,\ armed\_officer\}$ at state $(PB, 2)$ we have that:

$$Reject((PB, 2), M_{PB,2}) = \{carry\_weapon \leftarrow armed\_officer\}$$
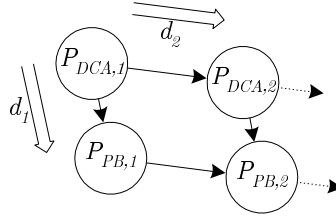$$Default\,(\mathcal{P}_{PB,2}, M_{PB,2}) = \{not\,exception\}$$

Fig. 2.

it is trivial to see that

$$M_{PB,2} = least\left([\mathcal{P}_{PB,2} - Reject((PB,2), M_{PB,2})] \cup Default\left(\mathcal{P}_{PB,2}, M_{PB,2}\right)\right)$$

which means that in spite of rule 2, since the exceptions have not been regulated yet, rule 1 prevails for all situations, and no one can carry a weapon aboard an airplane. This would correspond to the only stable model at state $(PB, 2)$.   □

   The applicability of $\mathcal{MDLP}$ in a multi-agent context is not limited to the assignment of a single semantics to the overall system, i.e., the multi-agent system does not have to be described by a single DAG. Instead, we could determine each agent's view of the world by associating a DAG with each agent, representing its own view of its relationships to other agents and of these amongst themselves. The stable models over a set of states from DAGs of different agents can provide us with interagent views.

Example 4. Consider a society of agents representing a hierarchically structured research group. We have the Senior Researcher (A_{sr}), two Researchers (A_{r1} and A_{r2}) and two students (A_{s1} and A_{s2}) supervised by the two Researchers. The hierarchy is deployed in Fig.3 a), which also represents the view of the Senior Researcher. Typically, students think they are always right and do not like hierarchies, so their view of the community is quite different. Fig.3 b) manifests one possible view by A_{s1}. In this scenario, we could use $\mathcal{MDLP}$ to determine and eventually compare A_{sr}'s view, given by the stable models at state sr in Fig.3 a), with A_{s1}'s view, given by the stable models at state $s1$ in Fig.3 b). If we assign the following simple logic programs to the five agents:

$$P_{sr} = \{a \leftarrow b\} \quad P_{s1} = \{not\ a \leftarrow c\} \quad P_{s2} = \{\} \quad P_{r1} = \{b\} \quad P_{r2} = \{c\}$$
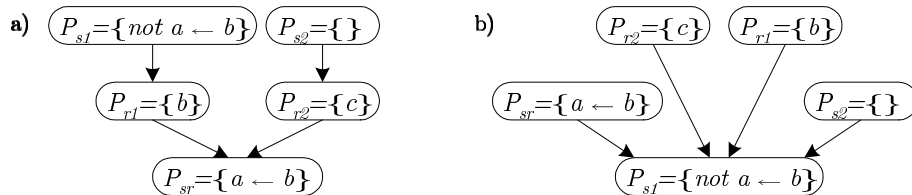

Fig. 3.

we have that state sr in Fig.3 a) has $M_{sr} = \{a, b, c\}$ as the only stable model, and state $s1$ in Fig.3 b) has $M_{s1} = \{not\,a, b, c\}$ as its only stable model. That is, according to student A_{s1}'s view of the world a is false, while according to the senior researcher A_{sr}'s view of the world a is true.   □

This example suggests $\mathcal{MDLP}$ to be a useful practical framework to study changes in behaviour of such multi-agent systems and how they hinge on the relationships amongst the agents, i.e. on the current DAG that represents them. $\mathcal{MDLP}$ offers a staple basic tool for the formal study of the social behaviour in multi-agent communities [10].

5 Transformational Semantics for MDLP

Definition 2 above establishes the semantics of $\mathcal{MDLP}$ by characterizing its stable models at each state. Next we present an alternative definition, based on a purely syntactical transformation that, given a *MDLP*, produces a generalized logic program whose stable models are in a one-to-one equivalence relation with the stable models of the *MDLP* previously characterized. The computation of the stable models at some state s reduces to the computation of the transformation followed by the computation of the stable models of the transformed program. This directly provides for an implementation of $\mathcal{MDLP}$ (publicly available at `centria.di.fct.unl.pt/~jja/updates`) and a means to study its complexity.

Without loss of generality, we extend the DAG D with an initial state (s_0) and a set of directed edges (s_0, s') connecting the initial state to all the sources of D. Similarly, if we wish to query a set of states, all needs doing is extending the *MDLP* with a new state α, as mentioned before, prior to the transformation. By $\overline{\mathcal{L}}$ we denote the language obtained from language $\mathcal{L}$ such that $\overline{\mathcal{L}} = \mathcal{L} \cup \{A^-, A_s, A_s^-, A_{P_s}, A_{P_s}^-, reject(A_s), reject(A_s^-) : A \in \mathcal{L}, s \in V \cup \{s_0\}\}$.

Definition 4 (Multi-dimensional Dynamic Program Update). *Let $\mathcal{P}$ be a MDLP, where $\mathcal{P} = (\mathcal{P}_D, D)$, $\mathcal{P}_D = \{P_v : v \in V\}$ and $D = (V, E)$. Given a fixed state $s \in V$, the multi-dimensional dynamic program update over $\mathcal{P}$ at state $\mathbf{s}$ is the generalized logic program $\boxplus_s\mathcal{P}$, which consists of the clauses below in the extended language $\overline{\mathcal{L}}$, where $D_s = (V_s, E_s)$ is relevancy DAG of D wrt s:*
    (RP) Rewritten program clauses:

$$A_{P_v} \leftarrow B_1, \ldots, B_m, C_1^-, \ldots, C_n^- \qquad A_{P_v}^- \leftarrow B_1, \ldots, B_m, C_1^-, \ldots, C_n^-$$

for any clause:

$$A \leftarrow B_1, \ldots, B_m, not\,C_1, \ldots, not\,C_n$$

respectively, for any clause:

$$not\,A \leftarrow B_1, \ldots, B_m, not\,C_1, \ldots, not\,C_n$$

in the program P_v, where $v \in V_s$.

(IR) Inheritance rules:

$$A_v \leftarrow A_u, not\, reject(A_u) \qquad A_v^- \leftarrow A_u^-, not\, reject(A_u^-)$$

for all atoms $A \in \mathcal{L}$ and all $(u, v) \in E_s$. The inheritance rules say that an atom A is true (resp. false) at state $v \in V_s$ if it is true (resp. false) at any ancestor state u and it is not rejected.

(RR) Rejection Rules:

$$reject(A_u^-) \leftarrow A_{P_v} \qquad reject(A_u) \leftarrow A_{P_v}^-$$

for all atoms $A \in \mathcal{L}$ and all $u, v \in V_s$ where $u < v$. The rejection rules say that if an atom A is true (resp. false) in the program P_v, then it rejects inheritance of any false (resp. true) atoms of any ancestor.

(UR) Update rules:

$$A_v \leftarrow A_{P_v} \qquad A_v^- \leftarrow A_{P_v}^-$$

for all atoms $A \in \mathcal{L}$ and all $v \in V_s$. Update rules state that atom A must be true (resp. false) at state $v \in V_s$ if it is made true (resp. false) in the program P_v.

(DR) Default Rules:

$$A_{s_0}^-$$

for all atoms $A \in \mathcal{L}$. Default rules describe the initial state s_0 by making all atoms false at that state.

(CS_s) Current State Rules:

$$A \leftarrow A_s \qquad A^- \leftarrow A_s^- \qquad not\, A \leftarrow A_s^-$$

for all atoms $A \in \mathcal{L}$. Current state rules specify the state s at which the program is being evaluated and determine the values of the atoms A, A^- and not A.

This transformation depends on the prior determination of the relevancy graph wrt. the given state. This choice was based on criteria of clarity and readability. Nevertheless this need not be so: one can instead specify declaratively, by means of a logic program, the notion of relevancy graph. As already mentioned, the stable models of the program obtained by the aforesaid transformation coincide with those characterized in Def.2, as expressed by the theorem:

Theorem 1. *Given a MDLP $\mathcal{P} = (\mathcal{P}_D, D)$, the generalized stable models of $\boxplus_s \mathcal{P}$, restricted to $\mathcal{L}$, coincide with the generalized stable models of $\mathcal{P}$ at state s according to Def.2.*

For lack of space, we do not present the proofs of the Theorems. In [11], the reader can find an extended version of this paper containing them.

6   Properties of MDLP

In this section we study some basic properties of $\mathcal{MDLP}$.

The next theorem states that adding or removing edges from a *DAG* of a *MDLP* preserves the semantics if the transitive closure of the two *DAGs* is the same DAG. In particular, it allows the use of a transitive reduction of the original graph to determine the stable models.

Theorem 2 (DAG Simplification). *Let $\mathcal{P} = (\mathcal{P}_D, D)$ be a MDLP, where $\mathcal{P}_D = \{P_v : v \in V\}$ and $D = (V, E)$. Let $\mathcal{P}_1 = (\mathcal{P}_D, D_1)$ be a MDLP, where $D_1 = (V, E_1)$ and $D^+ = D_1^+$. For any state $s \in V$, M is a* stable model *of $\mathcal{P}$ at state s iff M is a* stable model *of $\mathcal{P}_1$ at state s.*

One consequence of this theorem is that in order to determine the stable models at a set of states we only need to connect to the new node α the sinks of the relevancy DAG wrt. that set of states.

The following proposition relates the stable models of normal logic programs with those of *MDLP*s whose set of programs just contains normal logic programs.

Proposition 2. *Let $\mathcal{P} = (\mathcal{P}_D, D)$ be a MDLP, where $\mathcal{P}_D = \{P_v : v \in V\}$ and $D = (V, E)$. Let $S \subseteq V$ be a set of states and $D_S = (V_S, E_S)$ the relevancy DAG of D wrt. S. If all $P_v : v \in V_S$ are normal logic programs, then M is a* stable model *of $\mathcal{P}$ at states S iff M is a* stable model *of the (normal) logic program $\bigcup_{v \in V_S} P_v$*

The next theorem shows that $\mathcal{MDLP}$ generalizes its predecessor *DLP* [1].

Theorem 3 (Generalization of DLP). *Let $\mathcal{P}_D = \{P_s : s \in S\}$ be a DLP, i.e. a finite or infinite sequence of generalized logic programs, indexed by set of natural numbers $S = \{1, 2, 3, \ldots, n, ...\}$. Let $\mathcal{P} = (\mathcal{P}_D, D)$ be the MDLP, where $D = (S, E)$ is the acyclic digraph such that $E = \{(1, 2), (2, 3), ..., (n-1, n), ...\}$. Then, an interpretation M is a* stable model *of the dynamic program update (DLP) at state s, $\bigoplus_s \mathcal{P}_D$, if and only if M is a* stable model *of $\mathcal{P}$ at state s.*

Since *DLP* generalizes *Interpretation Updates*, originally introduced as *"Revision Programs"* by Marek and Truszczyński [15], then so does $\mathcal{MDLP}$. In [1], *DLP* was defined by means of a transformational semantics only. Theorems 1 and 3 establish Def. 2 as an alternative, declarative, characterization of *DLP*.

7   Conclusions and Future Work

We have introduced $\mathcal{MDLP}$ as a generalization of *DLP* in allowing for collections of states organized by arbitrary acyclic digraphs, and not just sequences of states. And therefore assigning semantics to sets and subsets of logic programs, on the basis of how they stand in relation amongst themselves, as defined by one acyclic digraph. Such a natural generalization imparts added expressiveness to updating, thereby amplifying the coverage of its application domains, as

we've tried to illustrate via some examples. The flexibility afforded by a DAG accrues to the scope and variety of possibilities. The new characteristics of multiplicity and composition of $\mathcal{MDLP}$ may be used to lend a "societal" viewpoint to *Logic Programming*. Application areas such as legal reasoning, software development, organizational decision making, multi-strategy learning, abductive planning, model-based diagnosis, agent architectures, and others, have already being successfully pursued by utilizing $\mathcal{MDLP}$.

Other frameworks exist for updates [20,18], and for combining logic programs via a partial order, developed for purposes other than updating. Namely, Disjunctive Ordered Logic ($\mathcal{DOL}$) [4], itself an extension of Ordered Logic, and $DLV^<$ [3], a language that extends LP with inheritance. Lack of space prevents us from elaborating on the comparison with these frameworks, so we defer to [5], where some considerations are made to that effect.

Some of the more immediate themes of ongoing work regarding the further development of $\mathcal{MDLP}$ comprise: allowing for the DAG itself to evolve by updating it with new nodes and edges; enhancing the $LUPS$ language to adumbrate update commands over DAGs; studying the conditions for and the uses of dropping the acyclicity condition; establishing a paraconsistent $\mathcal{MDLP}$ semantics and defining contradiction removal over DAGs.

References

1. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1-3):43–70, 2000. Abstract titled *Dynamic Logic Programming* appeared in Procs. of KR-98. 365, 367, 376

2. J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. Przymusinski. LUPS : A language for updating logic programs. *Artificial Intelligence.* To appear. 367

3. F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. In *Procs. of ICLP-99.* MIT Press, 1999. 377

4. F. Buccafurri, N. Leone, , and P. Rullo. Semantics and expressiveness of disjunctive ordered logic. *Annals of Math. and Artificial Intelligence*, 25(3-4):311–337, 1999. 377

5. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Considerations on updates of logic programs. In *Procs. of JELIA-00*, LNAI-1919. Springer, 2000. 377

6. M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In *Procs. of ICLP-88.* MIT Press, 1988. 367

7. K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998. 367

8. H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Procs. of KR-91.* Morgan Kaufmann, 1991. 365

9. E. Lamma, F. Riguzzi, and L. M. Pereira. Strategies in combined learning via logic programs. *Machine Learning*, 38(1/2):63–87, 2000. 365

10. J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In *Procs. of ATAL'01*, 2001. 374

11. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional logic programming. Technical report, Dept. Informatica, Universidade Nova de Lisboa, 2001. 375

12. J. A. Leite, F. C. Pereira, A. Cardoso, and L. M. Pereira. Metaphorical mapping consistency via dynamic logic programming. In *Procs. of AISB'00*. AISB, 2000. 365

13. J. A. Leite and L. M. Pereira. Generalizing updates: From models to programs. In *Procs of. LPKR-97*, volume 1471 of *LNAI*. Springer, 1997. 365

14. V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning (preliminary report). In *Procs. of KR-92*. Morgan-Kaufmann, 1992. 367

15. V. W. Marek and M. Truszczyński. Revision specifications by means of programs. In *Procs. of JELIA-94*, volume 838 of *LNAI*. Springer, 1994. 365, 376

16. Teodor C. Przymusinski and Hudson Turner. Update by means of inference rules. *Journal of Logic Programming*, 30(2):125–143, 1997. 365

17. P. Quaresma and I. P. Rodrigues. A collaborative legal information retrieval system using dynamic logic programming. In *Procs. of ICAIL-99*. ACM Press, 1999. 365

18. C. Sakama and K. Inoue. Updating extended logic programs through abduction. In *Procs. of LPNMR-99*. Springer, 1999. 377

19. Marianne Winslett. Reasoning about action using a possible models approach. In *Procs. of NCAI-88*. AAAI Press, 1988. 365

20. Y. Zhang and N. Foo. Updating logic programs. In *Procs. of ECAI'98*. Morgan Kaufmann, 1998. 377