The well supported semantics for multidimensional dynamic logic programs*

F. Banti[1], J. J. Alferes[1], A. Brogi[2], and P. Hitzler[3]

[1] CENTRIA, Universidade Nova de Lisboa, Portugal,
jja|banti@di.fct.unl.pt
[2] Dipartimento di Informatica, Università di Pisa, Italy,
brogi@di.unipi.it
[3] AIFB, Universität Karlsruhe, Germany,
hitzler@aifb.uni-karlsruhe.de

Abstract. *Multidimensional dynamic logic programs* are a paradigm which allows to express (partially) hierarchically ordered evolving knowledge bases through (partially) ordered multi sets of logic programs. They solve contradictions among rules in different programs by allowing rules in more important programs to reject rules in less important ones. This class of programs extends the class of *dynamic logic program* that provides meaning to *sequences of logic programs.* Recently the *refined stable model semantics* has fixed some counterintuitive behaviour of previously existing semantics for dynamic logic programs. However, it is not possible to directly extend the definitions and concepts of the refined semantics to the multidimensional case and hence more sophisticated principles and techniques are in order. In this paper we face the problem of defining a proper semantics for multidimensional dynamic logic programs by extending the idea of well supported model to this class of programs and by showing that this concept alone is enough for univocally characterizing a proper semantics. We then show how the newly defined semantics coincides with the refined one when applied to sequences of programs.

1 Introduction

Recently considerable effort has been devoted to explore the problem of how to update knowledge bases represented by logic programs (LPs) with new rules. This allows, for instance, to better use LPs for representing and reasoning with knowledge that evolves in time, as required in several fields of application. The LP updates framework has been used, for example, as the base of the MINERVA agent architecture [14] and the action description language EAPs [3].

Different semantics have been proposed [1, 2, 5, 6, 12, 15, 17, 19, 18] that assign meaning to arbitrary finite sequences $P_1, \ldots, P_m$ of logic programs, usually called *dynamic logic programs* (DyLPs), each program in the sequence representing a supervenient state of the world. The different states can be seen as

* This work was supported by project POSI/40958/SRI/01, FLUX, and by the European Commission within the 6th Framework P. project Rewerse, no. 506779.

representing different time points, in which case P_1 is an initial knowledge base, and the other P_is are subsequent updates of the knowledge base. The different states can also be seen as knowledge coming from different sources that are (totally) ordered according to some precedence, or as different hierarchical instances where the subsequent programs represent more specific information. The role of the semantics of DyLPs is to employ the mutual relationships among different states to precisely determine the meaning of the combined program comprised of all individual programs at each state. Intuitively, one can add, at the end of the sequence, newer rules or rules with precedence (arising from newly acquired, more specific or preferred knowledge) leaving to the semantics the task of ensuring that these added rules are in force, and that previous or less specific rules are still valid (by inertia) only as far as possible, i.e. that they are kept as long as they are not rejected. A rule is rejected whenever it is in conflict with a newly added one (*causal rejection of rules*). Most of the semantics defined for DyLPs [1, 2, 5, 6, 12, 15, 17, 19, 18] are based on such a concept of causal rejection. *Multidimensional dynamic logic programs* (MDyLPs) [13] generalize DyLPs by considering, instead of sequences, *partially ordered* multisets of programs. This generalization allows to combine in a single framework the possibility of having hierarchically ordered knowledge bases, with evolution in time. While most of the existing semantics[1] for DyLPs coincide on a large class of program updates (cf. [6, 11]), there are situations in which the set of (dynamic) stable models (SMs) differs from one semantics to the other. Usually such counter-examples show a counterintuitive behaviour of the semantics when dealing with particular kinds of recursive dependencies. Also the existing semantics for MDyLPs exhibit such counterintuitive behaviour as it emerges from the following example which also illustrates a possible usage of MDyLPs.

Example 1. Roughly speaking, a *joint venture* is a society of companies that is administrated by a delegated administrator. The administrator can pursue his own policy independent of the requests of the partners, though taking the directives of the partners into account. Suppose that two companies α and β constitute a joint venture j whose customers are other companies. The two companies give some directives to the administrator, represented by programs $P_{\alpha 1}$ and $P_{\beta 1}$. The administrator has his own policy which we represent by program P_{j1}. Moreover, both the companies and the administrator update their policies from time to time. We represent such updates, respectively, by programs $P_{\alpha t}$, $P_{\beta t}$ and P_{jt} where t is a time-stamp. Since the policy of the administrator has precedence over the directives of α and β, we assign precedence to the P_{js} over the other programs. At any given time t_k, the decisions of the administrator are described by the semantics of the MDyLP computed at the most recent node P_{jt}. The administrator respects a specific directive from a partner unless it con-

[1] In this paper we restrict our study to semantics generalizing the stable models semantics and that are based on causal rejection of rules. Semantics not based on causal rejection like the ones defined in [17, 19], or semantics which use more general forms of rejection like the one presented in [18] are outside the scope of this paper, and are only briefly mentioned in the conclusions.

flicts either with a more recent directive of the same partner or with any of the clauses representing the updates. For any order x, if $cost(x, z)$ is true then z represents the estimated expenses that the j should face in order to satisfy the commission x. Let us assume that the directive of β is to not decline any order whose cost is within a given limit C. This is encoded by the single rule program: $P_{\beta 1} : not\, decline(X) \leftarrow cost(X, Z), Z \leq C$. On the other hand, α assigns a degree of reliability to some customers and wants to decline any commission asked by a customer who is not reliable enough. So, $P_{\alpha 1}$ contains:

$$decline(X) \leftarrow comm(X, Y),\ reliability(Y, K),\ rel\_limit(Z),\ K \leq Z.$$

plus a database of facts $reliability(C, K)$ for customers C, and a fact $rel\_limit(li)$ for representing the limit under which a customer is considered to be unreliable. The initial policy of the administrator is to accept any commission that is acceptable and not declined, having also a set of rules for establishing when a commission is acceptable. Moreover, whenever several orders are received from the same costumer, if one order is accepted, then the others can not be declined. So, P_{j1} contains some rules for $acceptable(X)$ plus:

$$accept(X) \leftarrow acceptable(X), not\, decline(X). \tag{1}$$
$$not\, decline(X) \leftarrow accept(K), comm(X, Y), comm(K, Y). \tag{2}$$

Let us suppose a commission x_1 arrives from the company y, the commission is acceptable, and the reliability of y is high enough. Hence the order is accepted. At time t_2 the administrator updates his policy by deciding that no commission from the customer y will be declined. This is done by updating P_{j1} with the program P_{j2} with the rule: $not\, decline(X) \leftarrow comm(X, y)$.

At time t_3, α augments the limit of reliability. This is done by the update $P_{\alpha 3}$ with the two facts $not\, rel\_limit(li)$ and $rel\_limit(new\_li)$. Suppose that at time t_4 a commission x_2 arrives from y and the reliability of y is under the new limit of reliability. The directives in P_{α_1} say to decline the commission. Nevertheless the policy of the administrator encoded by the clause in P_{j2} conflicts with this directive. Since P_{j2} has precedence over $P_{\alpha 1}$, the latter is rejected and the order is accepted. Note how the partially ordered programs are used to represent precedence among rules coming from different sources, as well as to represent updates of rules.

So far we provided an example of how multidimensional dynamic logic programs work. We show now a problematic situation involving cyclic dependencies. At time t_5, another commission x_3 arrives from a new costumer y_2. The reliability of y_2 is below the current limit, but the estimated cost of x_3 is below the limit C and, moreover x_3 is acceptable. Hence the directives of α and β collide. Since the directives of the two partners are not comparable, none can overcome the other. From an intuitive point of view the policy of j should not allow to judge whether the order should be declined or not. Hence, intuitively the semantics of the considered $MDyLP$ should allow the administrator to detect the conflict and specify, by a new update, whether to decline the new order or accept it.

Unfortunately, none of the existing semantics for MDyLPs matches such intuition. The rules 1 and 2 have cyclic instances for $X = K$. Such a cycle is not relevant in the other examined cases. For $X = K = x_3$, instead, the rule 2 rejects the rule in $P_{\alpha 1}$ thus allowing the semantics to have a model where the commission x_3 is not declined.

It is clearly possible to find also much more complex examples involving complicated self dependencies among rules. Similar examples are known also in the case of DyLPs. In [1] the authors propose the *refined extension principle*,which should be satisfied by a proper semantics for DyLPs in order to avoid such counterintuitive behaviour and then proposed the *refined stable model semantics for DyLPs* that complies with such a principle. Unfortunately, as we show in section 3, the definition of the refined semantics cannot be extended directly to MDyLPs. Moreover, the refined extension principle is too weak for uniquely determining one "right" semantics. For example, the trivial semantics that assigns to each DyLP the empty set of models, satisfies the principle, which is obviously unsatisfactory. We hence need stronger new criteria and techniques.

We begin this paper by, after recalling preliminary notions in Section 2, providing insights of the existing semantics for DyLPs and MDyLPs and explain why a new approach to the problem, based on stronger criteria, is in order. Then, in Section 4 we introduce such a criterium by extending the notion of *well-supported model* (WS model) [4, 7] to DyLPs and MDyLPs. Fages [7] shows the equivalence between the concept of stable model and WS model, i.e. given a program P, an interpretation M is a stable model of P iff it is a WS model of P. By extending the definition of well supported model to MDyLPs, we obtain a new semantics for such class of programs. We also show how well supported models do not show counterintuitive behaviour in the illustrated example and, moreover, they provide new insights of the matrix of the behaviour of the other semantics for MDyLPs. Finally, we show that the well supported model semantics coincides with the refined semantics of [1] if we restrict to sequences of programs. For this reason we refer to the defined semantics also as the *refined semantics for MDyLPs*. Well-supported models do already provide a semantics for MDyLPs. Such a descriptive characterization, however, is not completely satisfactory for several reasons, the main one being the problem of finding a reasonable algorithm for its computation. So, in Section 5, we provide an alternative, though equivalent and more traditional characterization based on a fixpoint operator. We then establish relationships between the well supported semantics and the existing semantics for MDyLPs, and show that any WS model is also a model in the existing semantics.

2 Background and preliminaries

In the following, a *propositional language* $\mathcal{L}$ is a (possibly countably infinite) set of atoms. A *literal in* $\mathcal{L}$ is an atom A of $\mathcal{L}$ or the (default) negation $not\, A$ of an atom of $\mathcal{L}$. We say that A is the *default complement* of $not\, A$ and vice versa. Given a set of literals I, we say a literal L is *true* (resp. *false*) in I iff $L \in I$

(resp. $not\, L \in I$). In the sequel a (two-valued) *interpretation* is a set of literals of $\mathcal{L}$ such that for each atom $A \in \mathcal{L}$ *exactly one* of A or $not\, A$ belongs to I. To simplify the notation, whenever it is clear that we are talking about two-valued interpretations we omit all its negative literals. Let $\mathcal{L}$ and $\mathcal{L}'$ be two languages such that $\mathcal{L} \subset \mathcal{L}'$. Let M be an interpretation over $\mathcal{L}'$. We use the notation $M|_{\mathcal{L}}$ for the set of literals of M in $\mathcal{L}$. Given two interpretations M and M^* over $\mathcal{L}'$, we use the notation $M \equiv |_{\mathcal{L}} M^*$ for $M|_{\mathcal{L}} \equiv M^*|_{\mathcal{L}}$.

2.1 Well-supported models

The semantic analysis which we make in this paper rests on the notion of *level mapping* over a set of atoms $\mathcal{L}$, where a *level mapping* ℓ is a function from $\mathcal{L}$ to the set of natural numbers. We also lift ℓ to negative literals of the form $not\, A$, where A is an element of $\mathcal{L}$, by setting $\ell(not\ A) = \ell(A)$. Given a conjunction of literals $C = L_1, \ldots, L_n$ we further extend ℓ by assigning to C the value $\ell(L_i)$, where i is chosen such that the value of $\ell(L_i)$ is maximal, i.e. $\ell(C) = \max(\{\ell(L_i) : L_i \in C\})$. For convenience, and by slight abuse of notation, we assign the value -1 to the empty conjunction of literals. Our approach is stimulated by recent results on uniform characterizations of different semantics for LPs in terms of level mappings as introduced in [10] and extended in [9]. This perspective provides an additional tool and guidelines on how to obtain reasonable new semantics for new classes of programs.

A *normal logic program* over a language $\mathcal{L}$ is any (possibly countably infinite) set of rules of the form $A \leftarrow body$, where A is any atom of $\mathcal{L}$ and *body* is any conjunction of literals of $\mathcal{L}$. Several different (two-valued) semantics are being used for assigning meaning to programs, including the *supported model semantics* [4], the *minimal supported model semantics* [4] and the *stable model semantics* [8]. Given any program P, the set of all supported models ($SU(P)$), the set of all minimal supported models ($MSU(P)$) and the set of all stable models ($SM(P)$) of P are related by $SU(P) \supseteq MSU(P) \supseteq SM(P)$. For large classes of programs these sets of models coincide, but there are particular cases where the inclusions above are strict.

Example 2. Consider the program $P : \ A \leftarrow A$. P has unique SM $\emptyset$ which coincides with the unique minimal supported model. It has $\{A\}$ as a second supported model. All the cited semantics have $\emptyset$ as unique model of the program consisting of the empty set of clauses. Hence, for the supported model semantics adding tautologies of the form $A \leftarrow A$ to a program may change its semantics.

Example 3. Consider now $P_1 : A \leftarrow not\, A. \quad A \leftarrow A$. P_1 has no stable models but has $\{A\}$ as unique minimal supported model. The program $P_2 : \ A \leftarrow not\, A$, has no minimal supported model. Hence, again, the introduction of the tautology $A \leftarrow A$ has changed the semantics of the program.

Stable models for normal LPs can be characterized in terms of level mappings, and in this disguise they are termed *well-supported models* [7]. A model is well-supported iff it is possible to define a level mapping over the literals of the language, such that a literal A belongs to the model iff there is a rule in the program whose head is A, whose body is true in the considered model and the level of A is greater than the level of any atom in the body. Formally:

Definition 1. *Let P be a normal logic program over the language $\mathcal{L}$. An interpretation M over $\mathcal{L}$ is a* well-supported model of P *iff i) M is a model of P and ii) there exists a level mapping ℓ defined over $\mathcal{L}$, such that for each atom A in M there exists a rule $A \leftarrow A_1, \ldots, A_n, not\, B_1, \ldots, not\, B_m$ with $M \models A_1, \ldots, A_n, not\, B_1, \ldots, not\, B_m$ and $\ell(A) > \ell(Ai)$ for each A_i with $1 \leq i \leq n$.*

As formalized in the following result of [7], the WS models of a program P coincide with its stable models.

Theorem 1. *Let P be a normal logic program over $\mathcal{L}$. An interpretation M over $\mathcal{L}$ is a* well-supported model of P *iff it is a stable model of P.*

2.2 Semantics for DyLPs and MDyLPs

To represent negative information in logic programs and their updates, DyLPs use *generalized logic programs* (GLPs) [16], which allow for default negation not only in the premises of rules but also in their heads. A GLP over a language $\mathcal{L}$ is any (possibly countably infinite) set of rules of the form $L_0 \leftarrow L_1, \ldots, L_n$, where each L_i is a literal of $\mathcal{L}$. Given a rule τ as above, by $hd(\tau)$ we mean L_0 and by $B(\tau)$ we mean $\{L_1, \ldots, L_n\}$.

A *dynamic logic program* $\mathcal{P}$ over a language $\mathcal{L}$ is a finite sequence $P_1, \ldots, P_m$, where all the P_i's are GLPs over $\mathcal{L}$. We call the P_is *updates.* A *multidimensional dynamic logic program* $\mathcal{MP}$ is any partially ordered finite multiset of GLPs. Let $\mathcal{M}$ be a set of indices for the elements of $\mathcal{MP}$, and let $\prec$ be the partial order defined over $\mathcal{MP}$. For any index i, by P_i we denote the element of $\mathcal{MP}$ associated with i, and we call it an *update.* We often use the notation $i \prec j$ instead of $P_i \prec P_j$. Let τ and η be two rules appearing in $\mathcal{MP}$. As an abuse of notation, we also use the notation $\tau \prec \eta$ for denoting that τ and η belong, respectively, to the updates P_i and P_j with $P_i \prec P_j$. Two rules τ and η are said to be *not comparable* iff neither $\tau \prec \eta$ nor $\eta \prec \tau$ is true. For elements P_i and P_j of $\mathcal{M}$, we say that P_i is *less recent* than P_j iff $i \prec j$. Let P_n be an update of $\mathcal{MP}$. The *genealogy of* P_n, denoted by $\mathcal{MP}^n$, is the subset of the elements of $\mathcal{MP}$ which are less recent than P_j plus P_n itself. We use $\rho(\mathcal{P})$ to denote the multiset of all rules appearing in $\mathcal{MP}$ and $\rho(\mathcal{P}^n)$ to denote the multiset of all rules appearing in $\mathcal{MP}^n$. Note that, if the order defined over the MDyLP $\mathcal{MP}$ consisting of m elements is a *total order*, then $\mathcal{MP}$ is the DyLP $P_1, \ldots, P_m$ where P_i denotes the i^{th} element of $\mathcal{MP}$ such that $P_i \prec P_j$ iff $i < j$.

The *dynamic stable models semantics* for MDyLPs ($\mathcal{DS}$) is defined in [13] by assigning to each MDyLP a set of dynamic stable models. The basic idea of the semantics is that, if a later rule τ has a true body, then former rules in conflict with τ are *rejected.* Moreover, any atom A for which there is no rule with true body is considered false by default. The semantics is then defined by a fixpoint equation that, given an interpretation I, tests whether I has exactly the consequences obtained after removing from the multiset $\rho(\mathcal{P}^n)$ all the rejected rules, and imposing all the default assumptions given I. Formally:

Definition 2. *Let $\mathcal{MP}$ be any MDyLP over language $\mathcal{L}$, P_n be an update of $\mathcal{MP}$ and M be a two valued interpretation. Define*

$$\begin{aligned}
&Default(\mathcal{MP}^n, M) = \{not\, A \,|\, \nexists\, A \leftarrow body \in \rho(\mathcal{P}^n) : body \subseteq M\}\\
&Rej(\mathcal{MP}^n, M) = \{\tau \in P_i |\ \exists\, \eta \in P_j : i \prec j,\ \tau \bowtie \eta, B(\eta) \subseteq M\},
\end{aligned}$$

where $\tau \bowtie \eta$ means that τ and η are conflicting rules, i.e. the head of τ is the default complement of the head of η. Then M is a dynamic stable model *of $\mathcal{MP}$at P_n iff M is a fixpoint of $\Gamma^n_{\mathcal{MP}}$, defined by*

$$\Gamma^n_{\mathcal{MP}}(M) = least\,(\rho(\mathcal{P}^n) \setminus Rej(\mathcal{MP}^n, M) \cup Default(\mathcal{MP}^n, M))$$

where $least(P)$ denotes the least Herbrand model of the definite program obtained by considering each negative literal $not\, A$ in P as a new atom[2].

Other semantics for either DyLPs and MDyLPs based on causal rejection are the justified update ($\mathcal{JU}$) [15] and the update programs semantics ($\mathcal{UP}$) [6]. The latter is equivalent to the semantics proposed in [5]. All these semantics are extensions of the stable model semantics for normal and generalized LPs [8]; relations among them have been studied in [11, 12], and the main result is as follows. Given any MDyLP $\mathcal{MP}$, let $\mathcal{UP}(\mathcal{P}), \mathcal{JU}(\mathcal{P}), \mathcal{DS}(\mathcal{P})$ and be the set of models of $\mathcal{P}$ according to, respectively, the update programs, the justified update, the dynamic stable for MDyLPs. Then, as showed in [12, 11] the following (possibly strict) inclusions hold: $\mathcal{UP}(\mathcal{P}) \supseteq \mathcal{JU}(\mathcal{P}) \supseteq \mathcal{DS}(\mathcal{P})$ (3)

3 Some considerations on the existing semantics

As shown in [11] and [6], all the cited semantics for MDyLPs based on causal rejection coincide on large classes of programs. The examples where these semantics differ involve cyclic dependencies among rules, similar to the one illustrated in the example 1. As stated in the introduction, all the existing semantics for *MDyLPs* (wrongly) coincide on the program of example 1, i.e. they allow the program to have a model where the commission is not declined. Note that this is a consequence of the cycle introduced by the instances of the rules 1 and 2. In fact, the MDyLP obtained by removing the mentioned instances of the rules has no model according to any of the cited semantics. Indeed these two rules that should not affect the semantics allow somehow an undesired model.

Analogous behaviours are known also in the case of DyLPs as discussed in [1], where examples are given where cycles or even tautologies generate counterintuitive behaviour. In particular, all the semantics show counterintuitive behaviour

[2] Whenever clear from the context we will omit the $\mathcal{MP}$ in any of the above defined operators.

in cases where conflicting rules appear in the same update. In [1], the authors introduce the *refined extension principle* as a principle that any semantics should satisfy in order to avoid such behaviour. Moreover, the *refined dynamic stable model semantics* [1] or simply *refined semantics* for DyLPs is defined. Such semantics obeys the refined extension principle and, in fact, it avoids the mentioned counterintuitive behaviour. The definition of the refined semantics is the same of definition 2 but replacing the MDyLP $\mathcal{MP}$ with a DyLP $\mathcal{P}$ and replacing $i \prec j$ in the rejection operator by $i \leq j$. Intuitively, if two conflicting rules with true body and with heads, respectively, A and *notA* appear in the same update P_i they reject each other. Moreover they also reject all the previous rules with either head A and *not* A, and finally they also reject the default assumption *not* A. Hence, unless a new rule with head A or *not* A and true body appears in a more recent update, the considered program has no model because it is not possible to derive neither A nor *not* A. The refined semantics ($\mathcal{RS}$) for DyLPs is, among the cited semantics, the one that admits the less number of models. Hence, if we restrict to DyLPs, we can refine the inclusions 2.2 as follows $\mathcal{UP}(\mathcal{P}) \supseteq \mathcal{JU}(\mathcal{P}) \supseteq \mathcal{DS}(\mathcal{P}) \supseteq \mathcal{RS}(\mathcal{P})$ (4)

Unfortunately, the technique of mutual rejection of rules in the same state cannot be extended to the class of MDyLPs. In fact, if we extend directly the definition of refined model to MDyLPs, Example 1 still exhibits an undesired model. This originates from the fact that, in multidimensional case, unlike in the linear one, two rules may be *non comparable* even if they do not belong to the same update. If, instead we allow non comparable rules to reject each other, the obtained semantics allows too few models. Let us consider for instance the following program $\mathcal{MP}$ $P_1 : A$ $P_2 : not\,A$ $P_3 : A$ $P_4 : not\,A$ $P_1 \prec P_2$ $P_3 \prec P_4$ According to the proposal above, the rules in P_1 and P_4 reject each other, as well as the ones in P_2 and P_3. The default assumption is rejected as well and, as a result, we cannot derive neither A nor *not* A and hence $\mathcal{MP}$ would have no model. We regard this as counterintuitive, since, according to causal rejection, the facts A are rejected and $\{not\,A\}$ should be the unique model of the program. Moreover, this last counterintuitive behavior is not captured by the refined extension principle since this principle only ensures that, whenever it is satisfied, the introduction of particular kind of rules does not allow undesired models. Here the problem is exactly the opposite: the proposed semantics would have *too few* models. We conclude that it is simply not possible to *hack* the definition of the refined semantics in order to apply it to MDyLPs nor can we refer to the refined extension principle as a tool for univocally determine a semantics. What is needed is an entirely new methodology to attack the problem. The next section introduces such a methodology.

4 Well-supported models for MDyLPs

As we see from equation (4), the existing semantics for DyLPs can be ordered in a sequence where each semantics is a refinement of the previous one. Going right in the sequence, the conditions for an interpretation to be accepted as a model become stricter. It seems that research is trying to discard *bad models* from the semantics and to keep only *good models*. Is this really the case? Moreover, can we consider the sequence ended by the refined semantics, or do we need to further refine it? To answer these questions we need a formal definition of what we mean by *good model*. Recalling section 2.1, we can see an interesting historical parallelism between the evolution of semantics for DyLPs and the evolution of two-valued semantics for normal LPs, where the supported, minimal supported and stable model semantics are successive refinements. We also note similarities between examples 1 and 2, 3: in them, rules that should not play any semantic role change the behavior of some semantics by introducing more models. In the static case, this behaviour is rectified by the introduction of the concept of well-supported model. Guided by this historical perspective, we extend the notion of WS model to MDyLPs.

We first note that in the definition of well-supported models for normal LPs only the level of the positive literals in the body of a rule is considered. This happens because within normal LPs only positive literals are derived by rules, and the negative ones follow by negation by default. For DyLPs and MDyLPs, however, also negative literals can be derived by rules, since we allow negative literals in rule heads. More importantly, in the static case a rule plays a role only for deciding whether or not a given literal should be true or not. In the dynamic case, a rule is also used for *rejecting* other rules. Hence the concept of well-supportedness should be applied not only to the derivation of literals but also to the rejection of rules. More precisely, we require *well-supported rejection*: a rule can reject another rule in a previous update iff the body of the rejecting rule is true and the level of the body of the rejecting rule is less than the level of its head. The following definition formalizes this idea using level mappings.

Definition 3. *Let $\mathcal{MP}$ be any MDyLP over some language $\mathcal{L}$, let P_n be an update of $\mathcal{MP}$, ℓ be any level mapping over $\mathcal{L}$ and M be any two-valued interpretation over $\mathcal{L}$. Define the* set of rejected rules *w.r.t. ℓ by*[3]

$$\begin{aligned} Rej_\ell(\mathcal{MP}, M, n) = \{\tau \in P_i \mid \exists \eta \in P_j : i \prec j \preceq n, \tau \bowtie \eta, \\ M \models body(\eta), \ell(hd(\eta)) > \ell(B(\eta)) \ \}. \end{aligned}$$

Given the considerations above and Definition 3, it is now easy to extend the concept of WS model to MDyLPs. An interpretation M is a WS model if it is possible to find a level mapping such that: M is a model of all the rules that are not rejected w.r.t. the given level mapping and such that, for each atom A which is true in M, a non-rejected rule with head A exists, whose body is true and has level less than the level of A. The formal definition follows.

Definition 4. *Let $\mathcal{MP}$ be any MDyLP over some language $\mathcal{L}$ and let M be an interpretation over $\mathcal{L}$, P_n be an update of $\mathcal{M}$ and M be any interpretation over $\mathcal{L}$. We say that M is a* ***well-supported model at*** *P_n iff there exists a level mapping ℓ over $\mathcal{L}$ such that i) M is a model of $\rho(P) \setminus Rej_\ell(M,n)$ and ii)*

[3] Hereafter, we use the simplified notation $Rej_\ell(M, n)$ whenever this causes no ambiguity.

$\forall\ A \in M\ \exists\ \tau \in \rho(P) \setminus Rej_\ell(M,n)$ such that $hd(\tau) = A,\ \ \ell(A) > \ell(B(\tau))$ and τ is supported by M.

If we consider again the program of the example 1, we find that the model where the commission x_3 is not declined is not a well supported model. This also implies that the well supported model semantics for MDyLPs does not coincide with any of the existing semantics for such class of programs. If the partial order defined over the programs is a total one, then the considered program is a DyLP. In this case, it is possible to prove the analogue of Theorem 1.

Theorem 2. *Let $\mathcal{P}$ be any DyLP and M be an interpretation. Then M is a refined stable model of $\mathcal{P}$ iff it is a well-supported model of $\mathcal{P}$.*

For this reason, we refer to the defined semantics also as to the *the refined semantics for MDyLPs*. We have shown that the WS models coincide with the refined SMs for a DyLP.

Some comparisons of the refined semantics to other semantics for MDyLPs are in order. We would expect that any well supported model of a given program is also a model in any of the existing semantics for MDyLPs. In fact this result holds, as stated by the next theorem.

Theorem 3. *Let $\mathcal{MP}$ be any MDyLPs in the language $\mathcal{L}$, P_n be an update of $\mathcal{MP}$ and M be a well supported model of $\mathcal{MP}$ at P_n. Then M is also a model of $\mathcal{MP}$ in any of the semantics defined in [13, 5, 11].*

It is now easy to understand the different behaviour of the considered semantics. Two distinguished semantics differ for those cases when one semantics is a better approximation of the semantics of well-supported models than the other one.

5 Fixpoint characterization for well-supported models of MDyLPs

Well-supported models define a semantics for MDyLPs. However, this characterization is purely descriptive, which is obviously not entirely satisfactory for computational purposes. Moreover, to understand from this definition whether a given interpretation is a WS model of a given MDyLP, we have to face the problem of finding a corresponding level mapping or show that such a level mapping does not exist. This does not seem to be a reasonable approach for computing the semantics and, furthermore, may not lead to quick ways of testing whether a given interpretation is a well-supported model or not. For these reasons, we present an alternative characterization based on a fixpoint operator. We characterize our models as the fixpoints of an operator defined from interpretations to interpretations, in the spirit of the Gelfond-Lifschitz operator [8]. Given a program $\mathcal{MP}$ and an update P_n with index n, we associate with any pair $(\mathcal{MP}, n)$ an operator $\Gamma^S_{(\mathcal{MP},n)}$ as in Definition 7 below. We then obtain a characterization of the well-supported models of $\mathcal{MP}$ as the fixpoints of this operator.

The semantics of a MDyLP $\mathcal{MP}$ is defined with respect to the updates P_n of $\mathcal{MP}$. To establish the semantics consider only the genealogy of P_n. The underlying idea of the semantics is to collect in a single program all the rules of the given MDyLP and add new rules and predicates that specify whether a rule is rejected or not.

Let $\mathcal{MP}$ be a MDyLP over $\mathcal{L}$, P_j, P_k be programs of $\mathcal{MP}$ and $j \prec k$. If there exists a rule γ in P_k with head L, then every rule in P_j with head $not\, L$ could be rejected, depending on whether the body of γ is true or not. Formally:

Definition 5. *Let $\mathcal{MP}$ be an MDyLP over the language $\mathcal{L}$. Add new atoms $rej(L, i)$ not belonging to $\mathcal{L}$, where L is a literal in $\mathcal{L}$ and i ranges over the indices of the updates of $\mathcal{MP}$. The set of rejecting rules over the extended language is then defined as follows:*

$$Rj(\mathcal{MP}) = \{\ rej(not\, L, j)\ \leftarrow\ body.\ \mid L \leftarrow body. \in P_k\ \wedge\ j \prec k\ \}$$

Let $\mathcal{MP}$ be a multidimensional DyLP over $\mathcal{L}$, n be an index, L be a literal of $\mathcal{L}$, M be an interpretation over $\mathcal{L}$, P_i, P_j and P_k be programs of $\mathcal{MP}^n$ and τ^i and η^j be rules in, respectively P_i and P_j. We say that η^j *is a threat for L in* i, or alternatively that L *is threatened by* η^j iff the head of η^j is $not\, L$, its body is true in M, and $j \not\prec i$. We say that η^j is a threat for some other rule τ^i in P_i iff it is a threat for its head in i. A literal (rule) is considered to be *strictly safe* in P_i iff all its threats are rejected. Intuitively, derivations should only be made from safe rules. The main idea behind our definition is that a rule can be used to derive consequences iff it has already been established that the rule is safe. To achieve this result, we first consider the GLP given by the union of all the rules in $\mathcal{MP}$ with a new atom in the body of each rule which is satisfied only if the considered rule is safe. Then we introduce rules specifying which threats should be rejected in order to consider a literal (rule) as safe. Finally we introduce rules determining when a threat is rejected. Formally:

Definition 6. *Let $\mathcal{MP}$ be any multidimensional dynamic logic program in the language $\mathcal{L}$. Add new atoms $safe(L, i)$ not belonging to $\mathcal{L}$, where L is a literal in $\mathcal{L}$ and i ranges over the indices of the updates of $\mathcal{MP}$. We denote by $\Sigma(\mathcal{MP})$ the following set of rules.*

$$\Sigma(\mathcal{MP}) = \{L \leftarrow body, safe(L, i) | L \leftarrow body \in P_i\}$$

Let M be an interpretation over $\mathcal{L}$. The set of conditions for a literal L to be strictly safe at P_i is defined as follows.

$$cond^S(\mathcal{MP}, M, L, i) = \{rej(not\, L, j) \mid \exists \eta \in P_j, j \not\prec i, M \models B(\eta) \wedge \\ hd(\eta) = not\, L\}$$

By abuse of notation, $cond^S(\mathcal{MP}, M, L, i)$ also denotes the conjunction of all the literals in $cond^S(\mathcal{MP}, M, L, i)$. The set of strictly safe rules *is defined as*

$$Safe^S(\mathcal{MP}, M)) = \{safe(L, i) \leftarrow cond^S(\mathcal{MP}, M, L, i) | \exists\ \tau \in P_i :\ hd(\tau) = L\}.$$

Having specified the condition for a rule to be allowed to derive literals, we then add the set of default assumptions and compute the least model of the obtained program. Finally we discard the auxiliary literals computed.

Definition 7. *Let $\mathcal{MP}$ be any multidimensional dynamic logic program in the language $\mathcal{L}$, P_n be an update of $\mathcal{MP}$ and M be an interpretation over $\mathcal{L}$. We define the operator $\Gamma^S_{(\mathcal{MP},n)}$ on interpretations over $\mathcal{L}$ as follows.*

$$\begin{aligned}\Gamma^S_{(\mathcal{MP},n)}(I) = least\ (&\Sigma(\mathcal{MP}^n) \cup Safe^S(\mathcal{MP}^n, I) \cup \\ &Rj(\mathcal{MP}^n) \cup Default(\mathcal{MP}^n, I)\ )|_{\mathcal{L}}\end{aligned}$$

We are finally ready to define the refined semantics for MDyLPs. A *refined multi stable model* (RMSM) of an MDyLP at P_n is any fixpoint of the $\Gamma^S_{(\mathcal{MP},n)}$ operator.

Definition 8. *Let $\mathcal{MP}$ be any multidimensional dynamic logic program in the language $\mathcal{L}$, P_n be an element of $\mathcal{MP}$ and M be any interpretation over $\mathcal{L}$. We say that M is a* refined multi stable model *of $\mathcal{MP}$ at P_n iff $M = \Gamma^S_{(\mathcal{MP},n)}(M)$.*

The goal of Definition 8 is to provide a fixpoint characterization of the WS models of Definition 4. Indeed, this has been accomplished, as shown by the following theorem.

Theorem 4. *Let $\mathcal{MP}$ be any MDyLP in the language $\mathcal{L}$, n be an index and M be any two valued interpretation over $\mathcal{L}$. Then M is a* refined multi stable model *of $\mathcal{MP}$ at P_n iff M is a* well-supported model *of $\mathcal{MP}$ at P_n.*

6 Conclusions and future research

The initial purpose of the paper was to provide a semantics for MDyLPs based on causal rejection that can be properly considered to be *the* stable models-like semantics for such classes of programs. To obtain this, we extended the definition of well-supported model to the dynamic case. It turns out that, for DyLPs, our characterization coincides with the refined semantics. We provided also a fixpoint characterization of such semantics and established relationships between the new semantics and existing ones. Is it possible to conclude that the refined semantics is the proper extension of the SMs semantics to DyLPs and MDyLPs? Unfortunately there exists no theoretical result which is equivalent to the statement "*this is the correct semantics*". Nevertheless we claim that the characterizations given herein provide some evidence that further refinements of the semantics will not be necessary.

As mentioned in the introduction, there exist stable models-like semantics for LP updates [17, 19, 18] which are not (or at least not exclusively) based on the concept of causal rejection. Unlike those based on causal rejection, such semantics significantly differ from the refined semantics, and among each other, in properties, behaviors and underlying concepts. Comparisons of the semantics defined in [17, 19] can be found in [12]. Given the underlying differences, it is possible that in the future specific application areas will be found where different approaches to LP updates are needed for different applications. It is our strong opinion that DyLPs and MDyLPs can be a useful tool in several application areas, in particular in those areas related to web-oriented applications for AI where powerful reasoning capabilities have to be applied in highly dynamic environments and where merging knowledge from different sources is an important and challenging task.

References

1. J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. Semantics for dynamic logic programming: a principled based approach. In *LPNMR-7*. Springer, 2004.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, (1–3):43–70, 2000.
3. J.J. Alferes, F. Banti, and A. Brogi. From logic programs updates to action description updates. In J. Leite and P. Torroni, editors, *CLIMA V*, pages 227–242. Pre-Proceedings, 2004. ISBN: 972-9119-37-6.
4. K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *The Journal of Logic Programming*, 19–20:9–72, 1994.
5. F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. In D. De Schreye, editor, *ICLP-99*, 1999.
6. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On properties of semantics based on causal rejection. *Theory and Practice of Logic Programming*, 2:711–767, 2002.
7. François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *ICLP-5*, 1988.
9. P. Hitzler. Towards a systematic account of different logic programming semantics. In A. Günter, R. Kruse, and B. Neumann, editors, *KI2003*, 2003.
10. P. Hitzler and M. Wendt. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming*, 5(1–2):123–159, 2005.
11. M. Homola. Dynamic logic programming: Various semantics are equal on acyclic programs. In J. Leite and P. Torroni, editors, *CLIMA V*, pages 227–242. Pre-Proceedings, 2004. ISBN: 972-9119-37-6.
12. J. A. Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers in Artificial Intelligence and Applications.* IOS Press, 2003.
13. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional dynamic knowledge representation. In T. Eiter, M. Truszczynski, and W. Faber, editors, *LPNMR-01*, pages 365–378, 2001.
14. J. A. Leite, J. J. Alferes, and L. M. Pereira. Minerva – a dynamic logic programming agent architecture. In *Intelligent Agents VIII*, volume 2333 of *LNAI.* 2002.
15. J. A. Leite and L. M. Pereira. Iterated logic program updates. In *JICSLP-98*, 1998.
16. V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning. In B. Nebel, C. Rich, and W. Swartout, editors, *KR-92*, 1992.
17. C. Sakama and K. Inoue. Updating extended logic programs through abduction. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *LPNMR-99*, 1999.
18. J. Sefranek. A Kripkean semantics for dynamic logic programming. In *LPAR'2000.* Springer, 2000.
19. Y. Zhang and N. Y. Foo. Updating logic programs. In Henri Prade, editor, *ECAI-98*, 1998.