Layer Supported Models of Logic Programs

Luís Moniz Pereira and Alexandre Miguel Pinto
{lmp|amp}@di.fct.unl.pt

Centro de Inteligência Artificial (CENTRIA)
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract. For practical applications, the use of top-down query-driven proof-procedures is essential for an efficient use and computation of answers using Logic Programs as knowledge bases. Additionally, abductive reasoning on demand is intrinsically a top-down search method. A 2-valued semantics for Normal Logic Programs (NLPs) allowing for top-down query-solving is thus highly desirable.
The current standard 2-valued semantics for NLPs, the Stable Models (SMs) semantics, does not allow for top-down query-solving because it does not enjoy the relevance property — and moreover, it does not guarantee the existence of a model for every NLP. To overcome these current limitations we introduce here a new 2-valued semantics for NLPs — the Layer Supported Models semantics — which conservatively extends the SMs, enjoys relevance and cumulativity, guarantees model existence, and respects the Well-Founded Model. We also show how our semantics can be easily extended to deal with Disjunctive Logic Programs and Extended Logic Programs (including explicit negation), thus providing a practical, comprehensive and advantageous alternative to SMs-based Answer-Set Programming.
Keywords: Stable Models, Relevance, Semantics, Layering

1 Introduction

The semantics of Stable Models (SM) [7] is a cornerstone for the definition of some of the most important results in logic programming of the past two decades, providing an increase in logic programming declarativity and a new paradigm for program evaluation. When we need to know the 2-valued truth value of all the literals in a logic program for the problem we are modeling and solving, the only solution is to produce complete models. Depending on the intended semantics, in such cases, tools like *SModels* [11] or *DLV* [2] may be adequate because they can indeed compute whole models. However, the lack of some important properties of language semantics, like relevance, cumulativity and guarantee of model existence (enjoyed by, say, Well-Founded Semantics [6] (WFS)), somewhat reduces its applicability in practice, namely regarding abduction, creating difficulties in required pre- and post-processing. But WFS in turn does not produce 2-valued models, though these are often desired, nor guarantees 2-valued model existence.

Furthermore, with SM semantics, in an abductive reasoning situation, computing the whole model entails pronouncement about every abducible whether or not it is relevant to the problem at hand, and subsequently filtering the irrelevant ones. When we just want to find an existential answer to a query, we either compute a whole model and check if it entails the query (the way SM semantics does), or, if the underlying semantics we use enjoys the *relevance* property — which SM semantics do not — we can simply use a top-down proof-procedure (*à la* Prolog), and abduce by need. In this second case, the user does not pay the price of computing a whole model, nor the price of abducing all possible abducibles or their negations, and then filtering irrelevant ones, because the only abducibles considered will be those needed to answer the query.

SM semantics does not allow for top-down query-solving precisely because it does not enjoy the relevance property — and moreover, does not guarantee the existence of a model. Furthermore, frequently there is no need to compute whole models, like its implementations do, but just the partial models that sustain the answer to a query. Relevance would ensure these could be extended to whole models.

To overcome these limitations we developed a new 2-valued semantics for NLPs — the Layer Supported Models (LSM) — which conservatively extends the SMs, enjoys relevance, cumulativity, and guarantee of model existence, and respects the Well-Founded Model (WFM) [6]. The LSM semantics here presented builds upon the foundational step of the Layered Models semantics presented in [12] by now imposing a layered support requirement resulting in WFM consonance. In [12], neither layered support nor WFM respect are required. Intuitively, a program is conceptually partitioned into "layers" which are subsets of its rules. An atom is considered *true* if there is some rule for it at some layer, where all the literals in its body which are supported by rules of lower layers are also *true*. Otherwise that conclusion is false. That is, a rule in a layer must, to be useful, have the support of rules in the layers below. We also show how our semantics can be easily extended to deal with Disjunctive Logic Programs (DisjLPs) and Extended Logic Programs (ELPs, including explicit negation), thus providing a practical, comprehensive and advantageous alternative to SMs-based Answer-Set Programming.

The core reason SM semantics fails to guarantee model existence for every NLP is that the stability condition it imposes on models is impossible to be complied with by Odd Loops Over Negation (OLONs) [1]. In fact, the SM semantics community uses such inability as a means to write Integrity Constraints (ICs).

Example 1. **OLON as IC.** Indeed, using SMs, one would write an IC in order to prevent X being in any model with the single rule for some atom 'a': $a \leftarrow not\ a, X$. Since the SM semantics cannot provide a semantics to this rule whenever X holds, this type of OLON is used as IC. When writing such ICs under SMs one must be careful and make sure there are no other rules for a. But the really unintuitive thing about this kind of IC used under SM semantics is the meaning of the atom a. What does a represent?

The LSM semantics provides a semantics to all NLPs. ICs are implemented with rules for reserved atom $falsum$, of the form $falsum \leftarrow X$, where X is the body of the IC we wish to prevent being true. This does not prevent $falsum$ from being in some models. To avoid them, the user must either conjoin goals with $not\ falsum$ or, if inconsistency examination is desired, *a posteriori* discard such models. LSM semantics separates OLON semantics from IC compliance.

[1] OLON is a loop with an odd number of default negations in its circular call dependency path.

After notation and background definitions, we present the formal definition of LSM semantics and overview its properties. Conclusions and future work close the paper.

2 Background Notation and Definitions

Definition 1. ***Logic Rule.*** *A Logic Rule r has the general form*
$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$ *where H, the B_i and the C_j are atoms.*
We call H the head of the rule — also denoted by $head(r)$. And $body(r)$ denotes the set $\{B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m\}$ of all the literals in the body of r. Throughout this paper we will use 'not ' to denote default negation. When the body of the rule is empty, we say the head of rule is a fact and we write the rule just as H.

Definition 2. ***Logic Program.*** *A Logic Program (LP for short) P is a (possibly infinite) set of ground Logic Rules of the form in Definition 1.*
In this paper we focus mainly on NLPs, those whose heads of rules are positive literals, i.e., simple atoms; and there is default negation just in the bodies of the rules. Hence, when we write simply "program" or "logic program" we mean an NLP.

3 Layering of Logic Programs

The well-known notion of stratification of LPs has been studied and used for decades now. But the common notion of stratification does not cover all LPs, i.e., there are some LPs which are non-stratified.

Example 2. **Stratified vs Non-Stratified Programs.** Consider the following two programs P_1 and P_2. P_1 is stratified, according to the usual notion of stratification, whereas P_2 is not.

$$P_1 : \ x \leftarrow a \quad a \leftarrow not\ b \quad b \leftarrow not\ c \qquad P_2 : \ x \leftarrow a \quad a \leftarrow not\ b \quad b \leftarrow not\ a$$

Informally, in P_1, we say atom 'a' is in a stratum above of that of 'b', because there is a rule for 'a' with 'b' in its body; we say 'a' depends on 'b'. But in P_2 that dependency is symmetrical: 'b' also depends on 'a', and we cannot say if 'a' is in a stratum above of that of 'b' or vice-versa.

The usual syntactic notions of dependency are mainly focused on atoms. They are based on a dependency graph induced by the rules of the program. These notions are useful, and we include them below (defs. 3,4). However, for our purposes they are insufficient as they leave out important structural information about the call-graph of P. To cover that information we also define and present below the notion of a rule's dependency (defs. 5, 6). Indeed, layering puts rules in layers, not atoms.

Definition 3. ***Atom's direct dependency.*** *Atom B directly depends on atom A in P iff there is at least one rule with head B and with A or $not\ A$ in the body.*

Definition 4. ***Atom's dependency.*** *An atom's dependency is just the transitive closure of the atom's direct dependency.*

Definition 5. ***Rule's direct dependency.*** *Rule r directly depends on atom B iff any of B or $not\ B$ is in its body.*

Definition 6. ***Rule's dependency.*** *Rule r depends on atom B iff r directly depends on B or there is some X_i or $not\ X_i$ in its body such that atom X_i depends on B.*

Definition 7. ***Relevant part of*** P ***for atom*** A **–** $Rel_P(A)$**.** *It is the subset of rules of* P *with head* A *plus the set of rules of* P *whose heads the atom* A *depends on, cf. [3].*

Likewise for the relevant part for an atom A notion [3], we define and present the notion of relevant part for a rule r.

Definition 8. ***Relevant part of*** P ***for rule*** r **–** $Rel_P(r)$**.** *It is the set containing the rule* r *itself plus the set of rules relevant for each atom* r *depends on.*

Definition 9. ***Layering of a Logic Program*** P**.** *Given a logic program* P *a layering function* $L/1$ *is just any function defined over the rules of P, assigning each rule* $r = H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$ *a positive integer, such that:*

– $L(r) = max(L(r_{X_i}) : X_i = head(r_{X_i}))$ *if*
  r *directly depends on* X_i *and* X_i *depends on* H*, i.e.* r *is involved in a loop.*
– $L(r) \geq L(r_{B_i})$ *if*
  r *does not depend on* H*(i.e.,* r *is* not *involved in a loop) and*
  $max(L(r_{B_i}) : B_i = head(r_{B_i})) > max(L(r_{C_j}) : C_j = head(r_{C_j}))$
  i.e. some rule for a B_i *is in a layer above all rules for all* C_j*.*
– $L(r) > L(r_{C_j})$ *if*
  r *does not depend on* H*(*r *is* not *involved in a loop) and*
  $max(L(r_{B_i}) : B_i = head(r_{B_i})) \leq max(L(r_{C_j}) : C_j = head(r_{C_j}))$
  i.e. otherwise.

A layering of program P *is a partition* $P^1, \ldots, P^n$ *of* P *such that* P^i *contains all rules* r *having* $L(r) = i$*, i.e., those which depend only on the rules in the same layer or layers below it. This notion of layering does not correspond to any level-mapping [9], since the later is defined over atoms, and the former is defined over rules. Amongst the several possible layerings of a program* P *we can always find the least one, i.e., the layering with least number of layers and where the integers of the layers are smallest. Then fact rules are in layer 1. In the remainder of the paper when referring to the program's layering we mean such least layering (easily seen to be unique).*

In example 2 above, although P_2 has no stratification, it has a layering with a unique layer: P_2^1 is comprised of all three rules $a \leftarrow not\ b$, $b \leftarrow not\ a$, and $x \leftarrow a$.

Due to the definition of dependency, this definition of layer does not coincide with that of stratum for usual stratification [1], nor does it coincide with the layer definition of [13]. However, when the program at hand is stratified (according to [1]) it can easily be seen that its respective layering coincides with its stratification. In this sense, layering, which is always defined, is a generalization of the stratification. The original definition of stratification [1] was made on predicate names rather than atoms. By abandoning the restriction of a finite number of strata of [1], the definition of Local Stratification (that now applies to atoms) of Przymusinski [13] is obtained. It copes with infinite ground programs, such as: $a(X) \leftarrow not\ b(s(X))$ $\quad b(s(X)) \leftarrow not\ a(X)$

Still, whereas the ground instance of this program (assuming at least one constant symbol) is not locally stratified, its ground version comprises just one layer.

The layering of P is said to be depth-bound iff there is one "bottom" layer comprised of rules whose heads are not above any other literal, i.e., iff $P^1 \neq \emptyset$. In practice, all useful programs have a depth-bound layering, but for theoretical completeness we show that the LSM semantics — defined in the sequel — also deals with programs with depth-unbound layering. A typical case of a program with a depth-unbound layering (actually the only one with real theoretical interest, to the best of our knowledge) was presented by François Fages in [5]. We repeat it here for illustration and explanation.

Example 3. **Program with depth-unbound layering.**

$$p(X) \leftarrow p(s(X)) \qquad p(X) \leftarrow not\ p(s(X))$$

Ground (layered) version of this program, assuming there only one constant 0 (zero):

$$\begin{array}{rcl} p(0) \leftarrow p(s(0)) & \qquad & p(0) \leftarrow not\ p(s(0)) \\ p(s(0)) \leftarrow p(s(s(0))) & & p(s(0)) \leftarrow not\ p(s(s(0))) \\ p(s(s(0))) \leftarrow p(s(s(s(0)))) & & p(s(s(0))) \leftarrow not\ p(s(s(s(0)))) \\ \vdots \leftarrow \vdots & & \vdots \leftarrow \vdots \end{array}$$

The only layer supported model of this program is $\{p(0), p(s(0)), p(s(s(0)))\ldots\}$ or, in a non-ground form, $\{p(X)\}$. The theoretical interest of this program lies in that, although it has no OLONs it still has no SMs either because no rule is supported (under the usual notion of support), thereby showing there is a whole other class of NLPs to which the SMs semantics provides no model, due to the notion of support used.

4 Layer Supported Models Semantics

The Layer Supported Models semantics we now present is the result of the two new notions we introduced: the layering, formally introduced in section 3, which is a generalization of stratification; and the layered support, as a generalization of classical support. These two notions are the means to provide the desired 2-valued semantics which respects the WFM, as we will see below. Next we present the layered support notion, after we recap the classical support notion for easier comparison.

Definition 10. ***Classically Supported interpretation.*** *An interpretation M of P is classically supported iff every atom a of M is classically supported in M. Given a NLP P and some model M with atom $a \in M$, a is classically supported in M iff there is some rule r in P (where $r = a \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$) such that $M \models \{b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m\}$. This notion of support requires that* all *the literals in the body of some rule for a are* true *under M in order for a to be supported under M.*

Definition 11. ***Layer Supported interpretation.*** *An interpretation M of P is layer supported iff every atom a of M is layer supported in M. Given a NLP P and some model M with atom $a \in M$, a is layer supported in M iff*

$$\exists_{\substack{r \in P: \\ head(r)=a \\ \wedge L(r)=i}} M \models (body(r) \cap \{A, not\ A : \forall_{\substack{r' \in P: \\ head(r')=A}} L(r') < i\})$$

I.e., a is layer supported in M iff it has a rule whose body literals which are determined by rules of strictly lower layers are true in M. Otherwise, it follows that a is false. It is as if the unsupported rules in a layer are removed and CWA is applied to the atoms without any rules.

Theorem 1. ***Classical Support implies Layered Support.*** *Given a NLP P, an interpretation M, and an atom a such that $a \in M$, if a is classically supported in M then a is also layer supported in M.*

Proof. Trivial from the definitions of classical support and layered support. □

In programs without odd loops layered supported models are classically supported too.

Example 4. **Layered Unsupported Loop.** Consider program P:

$$c \leftarrow not\ a \qquad a \leftarrow c, not\ b \qquad b$$

The only rule for b is in the first layer of P. Since it is a fact it is always *true*. Knowing this, the body of the rule for a is *false* because unsupported (both classically and layered). Since it is the only rule for a, its truth value is *false*, and, consequently, c is *true*. This is the intuitively desirable semantics for P, which corresponds to its LSM semantics. LM and the LSM semantics differences reside both in their layering notion and the layered support requisite of Def. 11. In this example, if we used LM semantics, which does not exact layered support, there would be two models: $LM_1 = \{b, a\}$ and $LM_2 = \{b, c\}$. $\{b\}$ is the only minimal model for the first layer and there are two minimal model extensions for the second layer, as a is not necessarily false in the LM semantics because Def. 11 is not adopted. Lack of layered support lets LM semantics fail to comply with the WFM.

Definition 12. ***Layer Supported Model of*** P. *Let $P^1, \ldots, P^n$ be the least layering of P. A layer supported interpretation M is a Layer Supported Model of P iff*

$$\forall_{1 \leq i \leq n} M|_{\leq i} \text{ is a minimal layer supported model of } \bigcup_{1 \leq j \leq i} P^j$$

where $M|_{\leq i}$ denotes the restriction of M to heads of rules in layers less or equal to i:

$$M|_{\leq i} \subseteq M \cap \{head(r) : L(r) \leq i\}$$

The Layer Supported semantics of a program is just the intersection of all of its Layer Supported Models.

By definition, the minimal layer supported models up to and including a given layer, respect the minimal layer supported models up to the layers preceding it. This ensures the truth assignment to atoms in loops in higher layers is consistent with the truth assignments in loops in lower layers and that these take precedence in their truth labeling. As a consequence of the layered support requirement, layer supported models of each layer comply with the WFM of the layers equal to or below it. Combination of the (merely syntactic) notion of layering and the (semantic) notion of layered support makes the LSM semantics.

Layered support is a more general notion than that of perfect models [14], with similar structure. Perfect model semantics talks about “least models” rather than “minimal models” because in strata there can be no loops and so there is always a unique least model which is also the minimal one. Layers, as opposed to strata, may contain loops and thus there is not always a least model, so layers resort to minimal models, and these are guaranteed to exist (it is well known every NLP has minimal models).

It is worth noting that atoms with no rules and appearing in the bodies of some rule are necessarily "placed" in the lowest layer: an atom a having no rules is equivalent to having the single rule $a \leftarrow false$. Any minimal model of this layer will consider the heads of such rules to be false. This ensures compliance with the Closed World Assumption (CWA).

Example 5. **Atom with no rules.** Consider $P = \{a \leftarrow not\ b\}$. In this case the least layering of P assigns $L(b \leftarrow false)$=1 and $L(a \leftarrow not\ b) = 2$, and therefore $P^1 = \emptyset$ and $P^2 = \{a \leftarrow not\ b\}$. Necessarily, $M_1 = \emptyset$ (which means b is false, and says nothing about a), and $M_2 = \{a\}$. Notice although $\{b\}$ is a minimal model of P it is a non-minimal model of layer 1 and, hence, rejected by our LSMs definition.

Example 6. **Layered Unsupported Loop example revisited.** Consider P from Ex. 4. Clearly, b's rule is in P's first layer and P^1's unique minimal model is exactly $M_1 = \{b\}$. Now the second layer of P, i.e., P^2 has two minimal models: $M_{2_1} = \{a, b\}$, and $M_{2_2} = \{c, b\}$ — both are already including P^1's minimal model M_1. Among M_{2_1} and M_{2_2}, only M_{2_2} is layer supported; let us see why. Since a's rule and c's rule depend on each other they are in the same (second) layer of P. The only rule in a layer strictly below is b's. Consider M_{2_2}. Take the (only) rule for c ($c \leftarrow not\ a$) and intersect the set of literals in its body, $\{not\ a\}$, with the set of literals from atoms which are heads of rules strictly below c's, i.e., $\{b, not\ b\}$. The result is $\{not\ a\} \cap \{b, not\ b\} = \emptyset$. $M_{2_2} \models \emptyset$ trivially; hence M_{2_2} is layer supported. Now consider M_{2_1}. Take the (only) rule for a ($a \leftarrow c, not\ b$) and intersect the set of literals in its body, $\{c, not\ b\}$, with the set of literals from atoms which are heads of rules strictly below c's, i.e., $\{b, not\ b\}$. The result is $\{c, not\ b\} \cap \{b, not\ b\} = \{not\ b\}$. Now, $M_{2_1} = \{a, b\} \not\models \{not\ b\}$, hence M_{2_1} is not layer supported. Layer Supported Models give the intended semantics for P.

Example 7. **A joint vacation problem.** Three friends are planning a joint vacation. First friend says "I want to go to the mountains, but if that's not possible then I'd rather go to the beach". The second friend says "I want to go traveling, but if that's not possible then I'd rather go to the mountains". The third friend says "I want to go to the beach, but if that's not possible then I'd rather go traveling". However, traveling is only possible if the passports are OK. They are OK if they are not expired, and they are expired if they are not OK. We code this information as the following NLP:

$$\begin{array}{rlrl} beach \leftarrow & not\ mountain & pass\_ok \leftarrow & not\ exp\_pass \\ mountain \leftarrow & not\ travel & exp\_pass \leftarrow & not\ pass\_ok \\ travel \leftarrow & not\ beach, pass\_ok & & \end{array}$$

It is easy to see that the three rules on the left form an OLON over $beach$, $mountain$, and $travel$ and are in layer 2; and the rules for $pass\_ok$ and exp_pass are in layer 1. This program has only one SM: $\{exp\_pass, mountain\}$. But, looking at the rules relevant for $pass\_ok$ we find no irrefutable reason to assume exp_pass to be true. The LSM semantics allows $pass\_ok$ to be true yielding three other models besides the SM; those are: $LSM_1 = \{beach, mountain, pass\_ok\}$, $LSM_2 = \{beach, travel, pass\_ok\}$, and $LSM_3 = \{travel, mountain, pass\_ok\}$.

As proven in Theorem 5 in section 5, all SMs of a given program are also LSMs of it, thereby showing the LSM semantics is a conservative generalization of the SM semantics. Here, the $SM = \{exp\_pass, mountain\}$ is no exception: it is a minimal model of the program, and $\{exp\_pass\}$ is also a minimal model of layer 1. SMs, complying with the classical, more restrictive, notion of support, necessarily comply with the layered, slightly more relaxed (to afford all loops) notion of support. All other LSMs in the example are not SMs.

The first layer has two minimal models: $\{pass\_ok\}$ and $\{exp\_pass\}$. Assuming the first minimal model, the second layer has three minimal models which correspond to LSM_1, LSM_2, and LSM_3 above. Assuming the second minimal model (where exp_pass is true), the second layer has only one minimal model: the SM mentioned above $\{exp\_pass, mountain\}$ (which also a LSM). According to the LSM semantics, when the passports are OK, the three friends go on a joint vacation to any two of the three possible places.

Besides the lower layer atoms they depend on (if any), atoms involved in loops have no particular *raison d'être* in a model other than being part of a minimal model solution for the respective loop(s), i.e., their only support lies on lower layers. This is true for ELONs as well as OLONs. Thus, loops are just a way to write arbitrary disjunctive choices (viz. shifting rule of [4]). In this example there is no particular reason to choose $pass\_ok$ or exp_pass; we cannot say any of them to be supported for some reason. The same reasoning applies to the top layer where the OLON over $beach$, $mountain$, and $travel$ resides, provided that in the lower layer the truth of $pass\_ok$ has been adopted. The apparent lack of support of $beach$ in model $\{beach, mountain, pass\_ok\}$ is due to adopting the usual (classical) notion of support (where every atom true in a model must be supported by all the literals of a body of one of its rules), instead of adopting the new layered support (every atom true in a model must be classically supported just on the lower layers literals of a body of its rules).

The principle used by LSMs to provide semantics to any NLP — whether is has OLONs or not, whether it is depth-bound or not — is to accept all, and only, the minimal models that are layer supported, i.e., that respect the layers of the program. The principle used by SMs to provide semantics to some NLPs is just a "stability" (fixed-point) condition imposed on the SMs by the Gelfond-Lifschitz operator. This stability condition is too restrictive and it even gives rise to some incongruences (cf. example 8 below).

Example 8. **Even Loop Over Negation**[2] **vs Odd Loop Over Negation.** Consider P_1: $a \leftarrow not\ b \quad b \leftarrow not\ a$. It has two SMs: $SM_1 = \{a\}$, $SM_2 = \{b\}$. Now add the rules $a \leftarrow b$ and $b \leftarrow a$. The ELON is kept, but two OLONs appear now. The program now has no SMs, but it still has one $LSM = \{a, b\}$.

The example shows the incongruence in the SMs semantics when dealing with loops: it treats OLONs differently from ELONs and this incongruence stems from the stability requirement which, in our opinion, is too restrictive. The intended semantics of a loop over default negation, be it either an ELON or an OLON, be it written on purpose or be it produced by a series of updates or merges of different NLPs, is a disjunction. In example 8 above, the intended semantics of the ELON $a \leftarrow not\ b \quad b \leftarrow not\ a$ is,

[2] An Even Loop Over Negation (ELON), analogously to an OLON, is a loop in the dependency call graph with an intervening even (odd) number of default negations.

usually, $a \vee b$, and that is actually achieved by the SM semantics in this case. By adding the two rules $a \leftarrow b$ and $b \leftarrow a$, which do no more that stating an equivalence between a and b, the semantics of the resulting program should now be $(a \vee b) \wedge (a \Leftrightarrow b)$ which is just $a \wedge b$. The SM semantics fails in providing such behavior, whereas the LSMs semantics succeeds.

In the same manner of thinking, the intended semantics of program $a \leftarrow not\ b$ $b \leftarrow not\ c \quad c \leftarrow not\ a$ would be $(a \vee b) \wedge (b \vee c) \wedge (c \vee a)$; that is not achieved by the SM semantics. LSM semantics succeeds in doing so, while at the same time having upper layers' choices respect their lower layers' choices.

Extending the LSM semantics to Extended Logic Programs (where explicit negation is allowed, also in the heads of rules, besides default negation) is easily achieved by considering each explicitly negated literal $\neg L$ as just any another atom. Paraconsistent models may arise, but if unwanted can be discarded by semi-normalizing rules [10].

Extending the LSM semantics to Disjunctive LPs (allowing heads of rules to contain the disjunction of several atoms) is accounted for by the *shifting rule* program transformation [4, 8]. Example 9 is a clear illustration.

Example 9. **Map coloring.** Again, considering the SM semantics, the rules for any individual node of the classical problem of map coloring can be expressed as the following NLP (where we assume there are some facts for nodes and for the edges):

$$\begin{aligned} col(C, red) &\leftarrow not\ col(C, blue), not\ col(C, green) \\ col(C, blue) &\leftarrow not\ col(C, red), not\ col(C, green) \\ col(C, green) &\leftarrow not\ col(C, blue), not\ col(C, red) \end{aligned}$$

One can argue that there are OLONs here which, under the SM semantics, work as ICs preventing some undesired models. That is actually not the case in this situation: no OLON acts as an IC under SM semantics because every OLON has a symmetrical one (e.g, the OLON $col(C, red) \leftarrow not\ col(C, blue) \leftarrow not\ col(C, green) \leftarrow not\ col(C, red)$ is symmetrical to OLON $col(C, red) \leftarrow not\ col(C, green) \leftarrow not\ col(C, blue) \leftarrow not\ col(C, red)$) and both together form an ELON which is solvable by SM semantics. In fact, this is a typical example of an application of the *shifting rule* [4, 8] to transform a Disjunctive LP in to a NLP: the "original" Disjunctive LP would be $col(C, red) \vee col(C, blue) \vee col(C, green)$. In this example, since every SM is also a LSM, and there are no more minimal models besides the SMs, we conclude the LSM and SM semantics coincide.

5 Properties of the Layer Supported Models semantics

5.1 Existence

Theorem 2. ***Existence.*** *Every Normal Logic Program has a Layer Supported Model.*

Proof. By construction, it is always possible to find a layering for P and, therefore, its least layering. It is always possible to find a minimal layer supported model for layer 1 — for the first layer, a minimal layer supported model is just any minimal model since there are no layers below the first one. Moreover, for each layer $i+1$, it is alway possible to add to it the atoms in a minimal layer supported model of layer i as facts. Now, any minimal model of layer $i + 1$ with those facts is necessarily a minimal layer supported model of layer $i + 1$ including the minimal layer supported model of layer i. $\square$

Iterative method for building a Layer Supported Model For programs with depth-bound layering, a Layer Supported Model can be constructed in this way:

1. Find the layering of P, and consider $M_0 = \emptyset$
2. For any successor ordinals i, compute a minimal model M_{i+1} of $M_i \cup (P^{i+1}//M_i)$, where $P^{i+1}//M_i$ is obtained by:
   (a) Deleting from P^{i+1} all the rules with $not\ a$ in the body where $a \in M_i$
   (b) Deleting all a in the bodies of rules of P^{i+1} such that $a \in M_i$
   (c) Deleting from P^{i+1} all the rules with a in the body where there are no rules with head a in $P^{\leq i+1}$
   (d) Deleting all $not\ a$ in the bodies of rules of P^{i+1} such that there are no rules with head a in $P^{\leq i+1}$

   By construction, a minimal model M_{i+1} of such $M_i \cup (P^{i+1}//M_i)$ is a minimal layer supported model of P^{i+1}. This can easily be verified by noticing that all the rules in P^{i+1} with layered unsupported bodies are deleted when computing $P^{i+1}//M_i$ (steps (a), and (c)). Moreover, step (b) and (d) above only simplify the bodies of the remaining rules, respectively, by assuming *true* the atoms in M_i, and by enforcing the Closed World Assumption wrt. P^{i+1}. Steps (c) and (d) above are respectively called "Failure" and "Positive reduction" in [15]. Adding the atoms of M_i as facts guarantees that $M_{i+1} \supseteq M_i$ which is a necessary condition for M_{i+1} to be a minimal layer supported model.
3. For limit ordinals α, β, ω, $M_\omega = \bigcup_{\beta<\omega} M_\beta$ is a Layer Supported Model of $\bigcup_{\beta<\omega} P^\beta$ iff every M_β is a Layer Supported Model of $\bigcup_{\alpha<\beta} P^\alpha$.

5.2 Relevance

[3] presents definitions of the Relevance and Cumulativity properties of a semantics of logic programs. We recall them here for self containment.

Definition 13. ***Relevance.*** *A semantics for logic programs is said to be Relevant iff for every program P, $a \in Sem(P) \Leftrightarrow a \in Sem(Rel_P(a))$.*

Theorem 3. ***Relevance of LSM semantics.*** *The LSM semantics is relevant.*

Proof. According to definition 12, the LSM semantics of a program P — $LSM(P)$ — is the intersection of its LSMs. So, $a \in LSM(P) \Leftrightarrow \forall_{LSM_P(M)} a \in M$. For the LSM semantics the relevance property is expressed by $a \in LSM(P) \Leftrightarrow a \in LSM(Rel_P(a))$.

$\Rightarrow$: We assume $a \in LSM(P)$, so we can take any M such that $LSM_P(M)$ holds, and conclude $a \in M$. Assuming, by contradiction, that $a \notin LSM(Rel_P(a))$ then there is at least one LSM of $Rel_P(a)$ where a is *false*, i.e., $not\ a$ is *true*. Let M' be such a LSM of $Rel_P(a)$ where $a \notin M'$. Since we assumed $a \in M$, for every $LSM_P(M)$ we know there is some particular M such that $M \supseteq M'$; so $a \in M$ but $a \notin M'$. By definition M is a minimal model of P, so we know that in order for a to be in M it is necessary that there is either some rule of the form $a \leftarrow B$ or of the form $x \leftarrow B, not\ a$ in $P \setminus Rel_P(a)$. But every rule of the form $a \leftarrow B$ is, by definition, part of $Rel_P(a)$; so the only possibility is to have a rule $x \leftarrow B, not\ a$ in $P \setminus Rel_P(a)$. In this last case, since the LSMs are minimal models this means that a would be the unique atom satisfying the rule $x \leftarrow B, not\ a$, i.e., $M \models B$ and $M \not\models x$; but since LSMs are also layer supported this means that in such case a would not be in the model, whereas x would. And we would conclude that $a \notin M$ contradicting the initial hypothesis.

$\Leftarrow$: Assume $a \in LSM(Rel_P(a))$. Take the whole $P \supseteq Rel_P(a)$. Again, a will be in every LSM of P because a is in all LSMs of $Rel_P(a)$, and every LSM of P always contains one LSM of $Rel_P(a)$. $\square$

Relevance is the property that makes it possible to implement a top-down call-directed query-derivation proof-procedure — a highly desirable feature if one wants an efficient theorem-proving system that does not need to compute a whole model to answer a query. These methods are designed to try and identify whether a query literal belongs to some LSM, and to partially produce a LSM supporting a positive answer. The partial solution is guaranteed extendable to a full LSM because of relevance.

5.3 Cumulativity

Definition 14. ***Cumulativity.*** *A semantics is Cumulative iff for every program P*
$\forall_{a,b}(a \in Sem(P) \wedge b \in Sem(P)) \Rightarrow a \in Sem(P \cup \{b\})$

Theorem 4. ***Cumulativity of LSM semantics.*** *LSM semantics is cumulative.*

Proof. By definition 12, the semantics of a program P is the intersection of its LSMs. So, $a \in LSM(P) \Leftrightarrow \forall_{LSM_P(M)} a \in M$. For the LSM semantics cumulativity becomes expressed by $\forall_{a,b}(a \in LSM(P) \wedge b \in LSM(P)) \Rightarrow a \in LSM(P \cup \{b\})$

Let us assume $a \in LSM(P) \wedge b \in LSM(P)$. If b depends on a then there are $i \geq j$ such that $a \in M_j$ and $b \in M_i$, and $M \supseteq M_i \supseteq M_j$. It comes trivially that adding a as a fact to P does not change b's truth-value since every M_i including b already included a. If b does not depend on a's truth-value, and since the LSM semantics is relevant, b's truth-value will remain unchanged just by adding a as a fact to P. $\square$

5.4 Stable Models Extension

Theorem 5. ***Stable Models Extension.*** *Any Stable Model is an LSM of P.*

Proof. Assume M is an SM of P. It is well known that every SM is a minimal model, and moreover, that it is a *classically* supported model. From the definition of Layered Support, any classically supported model is necessarily also a Layer Supported Model. It follows from the definition of Layer Supported Model and these properties of Stable Models that every SM is also a LSM. $\square$

In example 7 we present a program with an SM — show it to be a LSM as well — and other non-SMs LSMs. Due to lack of space, the complexity analysis of this semantics is left out of this paper. Nonetheless, a brief note is due. Theorem 2 guarantees every NLP has at least one LSM, hence the complexity of finding if one LSM exists is trivial, when compared to SMs semantics. Since this semantics enjoys relevance, the complexity of brave reasoning (finding if there is any model of the program where some atom a is true) does not need to consider the whole P, but instead only $Rel_P(a)$. Nonetheless, it seems reasonable to assume that this is still an NP-hard task. Cautious reasoning (finding out if some atom a is in all models) boils down to finding if a is unconditionally *true* given its dependency graph.

5.5 Respect for the Well-Founded Model

Definition 15. ***Interpretation* M *of* P *respects the WFM of* P.** *An interpretation M respects the WFM of P iff M contains the set of all the* true *atoms of the WFM of P, and it contains no* false *atoms of the WFM of P.*

Theorem 6. ***Layer Supported Models respect the WFM.*** *Let P be an NLP, and $P^{\leq i}$ denote $\bigcup_{1 \leq j \leq i} P^j$. Each LSM $M|_{\leq i}$ of $P^{\leq i}$, where $M \supseteq M|_{\leq i}$, respects the WFM of $P^{\leq i} \cup M|_{<i}$.*

Proof. By definition, each $M|_{<i}$ is a full LSM of $P^{\leq i}$. Consider $P^{\leq 1}$. Any $M|_{\leq 1}$ contains the facts of P, and their direct positive consequences, since the rules for all of these are necessarily placed in the first layer in the least layering of P. Hence, $M|_{\leq 1}$ contains all the *true* atoms of the WFM of $P^{\leq 1}$. Layer 1 also contains whichever loops that do not depend on any other atoms besides those which are the heads of the rules forming the loop. Any such loops having no negative literals in the bodies are deterministic and, therefore, the heads of the rules forming the loop will be all *true* or all *false* in the WFM of $P^{\leq 1}$, depending on whether the bodies are fully supported by facts in the same layer, or not, and — in the latter case — if the rules are not involved in other types of loop making their heads undefined. In any case, an LSM of this layer will by necessity contain all the *true* atoms of the WFM of $P^{\leq 1}$. On the other hand, assume there is some atom b which is *false* in the WFM of $P^{\leq 1}$. b being *false* in the WFM means that either b has no rules or that every rule for b has an unsatisfiable body in $P^{\leq 1}$. In the first case, by definition 12 we know that b cannot be in any LSM. In the second case, every unsatisfiable body is necessarily unsupported, both classically and layered. Hence, b cannot be in any LSM of $P^{\leq 1}$. This means that any LSM contains no atoms *false* in the WFM of $P^{\leq 1}$. By definition $M|_{\leq i+1}$ is an LSM of $P^{\leq i+1}$ iff $M|_{\leq i+1} \supseteq M|_{\leq i}$, for some LSM $M|_{\leq i}$ of $P^{\leq i}$, which means the LSMs $M|_{\leq i+1}$ of $P^{\leq i+1}$ are exactly the LSMs $M|_{\leq i+1}$ of $P^{\leq i+1} \cup M|_{\leq i}$. Adding the $M|_{\leq i}$ atoms as facts imposes them as *true* in the WFM of $P^{\leq i+1} \cup M|_{\leq i}$. The then deterministically *true* consequences of layer $i+1$ — the *true* atoms of the WFM of $P^{\leq i+1} \cup M|_{\leq i}$ — become necessarily present in every minimal model of $P^{\leq i+1} \cup M|_{\leq i}$, and therefore in its every LSM $M|_{\leq i+1}$. On the other hand, every atom b *false* in the WFM of $P^{\leq i+1} \cup M|_{\leq i}$ has now unsatisfiable bodies in all its rules (up to this layer $i+1$). Hence, b cannot be in any LSM of $P^{\leq i+1} \cup M|_{\leq i}$. Therefore, every $M|_{\leq i+1}$ respects the WFM of $P^{i+1} \cup M|_{\leq i}$. □

6 Conclusions and Future Work

We have defined the LSM semantics for *all* NLPs, complying with expressed requirements: 2-valued semantics, extending SMs, guarantee of model existence, relevance and cumulativity, and respecting the WFM. We have also seen how this semantics can be easily extended to cater of ELPs and Disjunctive LPs. OLONs no longer act as ICs, be they intentionally written with that purpose or not. Instead, ICs are written as just $\bot \leftarrow IC\_Body$; and the user should conjoin $not\ \bot$ with her query if consistency is required (this can, of course, be automated).

The applications afforded by LSMs are all those of SMs plus those requiring OLONs for model existence, and those where OLONs are actually employed for problem representation. The guarantee of model existence is essential in applications where knowledge sources are diverse (like in the semantic web), and where the bringing together of such knowledge (automated or not) can give rise to OLONs that would otherwise prevent the resulting program from having a semantics, thereby brusquely terminating the application. A similar situation can be brought about by self- and mutually-updating programs, including in the learning setting, where unforeseen OLONs would stop short an ongoing process if the SM semantics were in use. Hence, apparently there is only to gain in exploring the adept move from SM semantics to the more general LSM one.

Work under way concerns the efficient implementation of the LSM semantics.

References

1. K.R. Apt and H.A. Blair. Arithmetic classification of perfect models of stratified programs. *Fundam. Inform.*, 14(3):339–343, 1991.
2. S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv system: Model generator and advanced frontends (system description). In *Workshop in Logic Programming*, 1997.
3. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: I, II. *Fundamenta Informaticae*, XXII(3):227–255, 257–288, 1995.
4. J. Dix, G. Gottlob, V.W. Marek, and C. Rauszer. Reducing disjunctive to non-disjunctive semantics by shift-operations. *Fundamenta Informaticae*, 28:87–100, 1996.
5. F. Fages. Consistency of Clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1:51–60, 1994.
6. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. of ACM*, 38(3):620–650, 1991.
7. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
8. M. Gelfond, H. Przymusinska, V. Lifschitz, and M. Truszczynski. Disjunctive defaults. In *KR-91*, pages 230–237, 1991.
9. P. Hitzler and M. Wendt. A uniform approach to logic programming semantics. *TPLP*, 5(1-2):93–121, 2005.
10. T. Janhunen. Classifying semi-normal default logic on the basis of its expressive power. In *Procs. LPNMR'99*, volume 1730 of *LNAI*, pages 19–33. Springer Verlag, 1999.
11. I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Procs. LPNMR'97*, volume 1265 of *LNAI*, pages 420–429, July 1997.
12. L.M. Pereira and A.M. Pinto. Layered models top-down querying of normal logic programs. In *Procs. PADL'09*, volume 5418 of *LNCS*, pages 254–268. Springer, January 2009.
13. T. C. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model. In *PODS*, pages 11–21. ACM Press, 1989.
14. T.C. Przymusinski. Perfect model semantics. In *ICLP/SLP*, pages 1081–1096, 1988.
15. U. Zukowski, B. Freitag, and S. Brass. Transformation-based bottom-up computation of the well-founded model. In *Non-Monotonic Extensions of Logic Programming (NMELP'96)*, volume 1216 of *LNAI*, pages 171–201. Springer, 2001.