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. Building upon the 2-valued Layered Models semantics for normal programs, we introduce a refinement — the Layer Supported Models semantics — which, besides keeping all of LMs' properties, furthermore respects the Well-Founded Model.
Keywords: Stable Models, Relevance, Semantics, Layering

1 Introduction

In [5] we presented the Layered Models semantics, a 2-valued semantics for Normal Logic Programs (NLPs) which guarantees model existence, enjoys relevance and cumulativity, and is a conservative extension of the Stable Models (SMs) semantics [3]. Although the LMs proposed in [5] already enforces minimality of its models, it does not ensure compatibility with the Well-Founded Model (WFM). The refinement to the LMs we now propose — Layer Supported Models (LSMs) semantics — obeys the proviso that each model respects the WFM. To achieve this we simply refine the notion of layering original of [5]. Intuitively, a program is conceptually partitioned into "layers" which are subsets of its rules. Rules forming loops are placed in the same layer, while rules not forming loops are placed in different layers — higher layer rules depending on lower layer rules, but not vice-versa. An atom is then considered *true* in a model if there is some rule for it, at some layer, where all literals in its body supported by rules of lower layers are also *true*. Otherwise the atom is *false*.

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) — like $a \leftarrow not\ a, X$ [1]. In fact, the SM semantics community uses such inability as a means to write Integrity Constraints (ICs).

The LSM semantics provides a semantics to all NLPs. In the $a \leftarrow not\ a, X$ example above, whenever X is *true*, the only LSM is $\{a, X\}$. For LSM semantics OLONs are not ICs. 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. From a theoretical standpoint this means that the LSM semantics does not include an IC compliance enforcement mechanism. ICs must be dealt with in two possible ways: either by 1) a syntactic post-processing step, as a "test phase" after the model generation "generate phase"; or by 2) embedding the IC compliance in a query-driven (partial) model computation, where such method can be a

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

top-down query-solving one *a la* Prolog, since the LSM semantics enjoys relevance. In this second case, the user must conjoin query goals with $not\ falsum$. If inconsistency examination is desired, like in the 1) case above, models including $falsum$ can be discarded *a posteriori*. This is how LSM semantics separates OLON semantics from IC compliance.

After notation and background definitions, we present the formal definition of LSM semantics and overview its properties. 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. Work under way concerns the efficient implementation of the LSM semantics.

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. 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. 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. Useful these notions might be, 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 below the notion of a rule's dependency. Indeed, layering puts rules in layers, not atoms. An 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. An atom's dependency is just the transitive closure of the atom's direct dependency. A rule directly depends on atom B iff any of B or $not\ B$ is in its body. A rule's dependency is just the transitive closure of the rule's direct dependency. The Relevant part of P for some atom A, represented by $Rel_P(A)$, 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. [2]. Likewise for the relevant part for an atom A notion [2], we define and present the notion of relevant part for a rule r. The Relevant part of P for rule r, represented by $Rel_P(r)$, is the set containing the rule r itself plus the set of rules relevant for each atom r depends on.

Definition 2. ***Parts of the body of a rule.*** *Let $r = H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$ be a rule of P. Then, $r^l = \{B_i, not\ C_j : B_i$ depends on $H \wedge C_j$ depends on $H\}$. Also, $r^B = \{B_i : B_i \in (body(r) \setminus r^l)\}$, and $r^C = \{C_j : not\ C_j \in (body(r) \setminus r^l)\}$.*

Definition 3. ***HighLayer function.*** *The $HighLayer$ function is defined over a set of literals: its result is the highest layer number of all the rules for all the literals in the set, or zero if the set is empty.*

Definition 4. ***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', where P' is obtained from P by adding a rule of the form $H \leftarrow falsum$ for every atom H with no rules in P, assigning each rule $r \in P'$ a positive integer, such that:*

- $L(r) = 0$ *if* $falsum \in body(r)$*, otherwise*
- $L(r) \geq max(HighLayer(r^l), HighLayer(r^B), (HighLayer(r^C) + 1))$

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 [4], since the later is defined over atoms, and the former is defined over rules. Also, due to the definition of dependency, layering does not coincide with stratification [1], nor does it coincide with the layer definition of [6]. 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.

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. In the remainder of the paper when referring to the program's layering we mean such least layering (easily seen to be unique).

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.

An interpretation M of P is classically supported iff every atom a of M is classically supported in M, i.e., *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 5. ***Layer Supported interpretation.*** *An interpretation M of P is layer supported iff every atom a of M is layer supported in M, and this holds iff a has a rule r where all literals in $(body(r) \setminus r^l)$ are true in M. Otherwise, it follows that a is false.*

Theorem 1. ***Classical Support implies Layered Support.*** *Given an 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 1. **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* in the WFM. 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* in the WFM, and, consequently, c is *true* in the WFM. 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. 5. 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. 5 is not adopted. Lack of layered support lets LM semantics fail to comply with the WFM. Note that adding a rule like $b \leftarrow a$ would not affect the semantics of the program, according to LSM. This is so because, such rule would be placed in the same layer with the rules for a and c, but leaving the fact rule b in the strictly lower layer.

Intuitively, the minimal layer supported models up to and including a given layer, respect the minimal layer supported models up to the layers preceding it. It follows trivially that layer supported models are minimal models, by definition. 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.

Definition 6. ***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*

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

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.

Layered support is a more general notion than that of perfect models [7], 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 $a \leftarrow falsum$. 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).

In [5] the authors present an example (7) where three alternative joint vacation solutions are found by the LM semantics, for a vacation problem modeled by an OLON. The

solutions found actually coincide with those found by the LSM semantics. We recall now a syntactically similar example (from [8]) but with different intended semantics, and show how it can be attained in LSM by means of ICs.

Example 2. **Working example [8].**

$$tired \leftarrow not\ sleep \qquad sleep \leftarrow not\ work \qquad work \leftarrow not\ tired$$

As in the example 7 of [5], the LSM semantics would provide three solutions for the above program: $\{work, tired\}, \{work, sleep\}, \{sleep, tired\}$. Although some (or even all!) of these solutions might be actually plausible in a real world case, they are not, in general, the intended semantics for this example. With the LSM semantics, the way to prune away some (or all) of these solutions is by means of ICs. For example, to eliminate the $\{work, sleep\}$ solution we would just need to add the IC $falsum \leftarrow work, sleep$.

The principle used by LSMs to provide semantics to any NLP — whether with OLONs 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 only some NLPs is a "stability" (fixed-point) condition imposed on the SMs by the Gelfond-Lifschitz operator.

5 Properties of the Layer Supported Models semantics

Although the definition of the LSMs semantics is different from the LMs of [5], this new refinement enjoys the desirable properties of the LMs semantics, namely: guarantee of model existence, relevance, cumulativity, and being a conservative extension of the SMs semantics. Moreover, and this is the main contribution of the LSMs semantics, it respects the Well-Founded Model.

Due to lack of space, the complexity analysis of this semantics is left out of this paper. Nonetheless, a brief note is due. Model existence is guaranteed for every NLP, hence the complexity of finding if one LSM exists is trivial, when compared to SMs semantics. Brave reasoning — finding if there is any model of the program where some atom a is true — is an intrinsically NP-hard task from the computational complexity point of view. But since LSM semantics enjoys relevance, the computational scope of this task can be reduced to consider only $Rel_P(a)$, instead of the whole P. From a practical standpoint, this can have a significant impact in the performance of concrete applications. Cautious reasoning (finding out if some atom a is in all models) in the LSM semantics should have the same computational complexity as in the SMs, i.e., coNP-complete.

5.1 Respect for the Well-Founded Model

Definition 7. ***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 2. ***Layer Supported Models respect the WFM.*** *Let P be an NLP, and $P^{\leq i}$ denote $\bigcup_{1 \leq j \leq i} P^j$, where P^j is P's j layer. Each LSM $M|_{\leq i}$ of $P^{\leq i}$, where $M \supseteq M|_{\leq i}$, respects the WFM of $P^i \cup M|_{<i}$.*

Proof. By hypothesis, each $M|_{\leq i}$ is a full LSM of $P^{\leq i}$. Consider P^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^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^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^1. On the other hand, assume there is some atom b which is *false* in the WFM of P^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^1. In the first case, by definition 6 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^1. This means that any LSM contains no atoms *false* in the WFM of P^1, and, therefore, that they must respect the WFM of P^1.

By hypothesis $M|_{\leq i+1}$ 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 $P^{i+1} \cup M|_{\leq i}$. Adding the $M|_{\leq i}$ atoms as facts imposes them as *true* in the WFM of $P^{i+1} \cup M|_{\leq i}$. The then deterministically *true* consequences of layer $i+1$ — the *true* atoms of the WFM of $P^{i+1} \cup M|_{\leq i}$ — become necessarily present in every minimal model of $P^{i+1} \cup M|_{\leq i}$, and therefore in its every LSM $M|_{\leq i+1}$. On the other hand, every atom b *false* in the WFM of $P^{i+1} \cup M|_{\leq i}$ has now unsatisfiable bodies in all its rules (up to this layer $i+1$). Hence, b cannot be in any LSM of $P^{i+1} \cup M|_{\leq i}$. Therefore, every $M|_{\leq i+1}$ respects the WFM of $P^{i+1} \cup M|_{\leq i}$. Hence, more generally, every $M|_{\leq i}$ respects the WFM of $P^i \cup M|_{<i}$

□

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. J. Dix. A Classification-Theory of Semantics of Normal Logic Programs: I, II. *Fundamenta Informaticae*, XXII(3):227–255, 257–288, 1995.
3. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
4. P. Hitzler and M. Wendt. A uniform approach to logic programming semantics. *TPLP*, 5(1-2):93–121, 2005.
5. 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.
6. 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.
7. T.C. Przymusinski. Perfect model semantics. In *ICLP/SLP*, pages 1081–1096, 1988.
8. Teodor Przymusinski. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187, 1994.