First-Order Encodings for Modular Nonmonotonic Datalog Programs*

Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9–11, A-1040 Vienna, Austria
{dao,eiter,fink,tkren}@kr.tuwien.ac.at

Abstract. Recently Modular Nonmonotonic Logic Programs (MLP) have been introduced which incorporate a call-by-value mechanism and allow for unrestricted calls between modules, including mutual and self recursion, as an approach to provide module constructs akin to those in conventional programming in Nonmonotonic Logic Programming under Answer Set Semantics. This paper considers MLPs in a Datalog setting and provides characterizations of their answers sets in terms of classical (Herbrand) models of a first-order formula, extending a line of research for ordinary logic programs. To this end, we lift the well-known loop formulas method to MLPs, and we also consider the recent ordered completion approach that avoids explicit construction of loop formulas using auxiliary predicates. Independent of computational perspectives, the novel characterizations widen our understanding of MLPs and they may prove useful for semantic investigations.

1 Introduction

Since the early days of Datalog, modularity aspects have been recognized as an important issue, and already the seminal notion of stratification [1] builds on an evaluation of subprograms in an ordered way. This has been later largely elaborated to notions like modular stratification [20] and XY-stratification incorporated in the $\mathcal{LDL}$++ system [2], and has been generalized to a syntactic notions of modularity for disjunctive Datalog programs [7, 9] that, in the context of non-monotonic logic programming, has been independently found as Splitting Sets [18]. More recently, research on modularity where in contrast subprograms may mutually depend on each other has been intensified, with DLP-functions [15] being the most prominent example to provide a Gaifman-Shapiro-style module architecture [13].

However the above concepts do not cater a module concept as familiar in conventional imperative and object-oriented languages, where procedures come with parameters that are passed on during the evaluation. To provide support for this, [8] developed modular logic programs, based on an extension of logic programs with genuine generalized quantifiers, where modules can receive input that is passed on in a call-by-value mode, in addition to the usual call-by-reference access to atoms in other modules. Limitations of this seminal approach have been recently overcome with an generalized and semantically

* This research has been supported by the Austrian Science Fund (FWF) project P20841 and the Vienna Science and Technology Fund (WWTF) project ICT-08 20.

refined notion of Modular Nonmonotonic Logic Programs (MLPs) in [6] under the answer set semantics [14].

Roughly, an MLP is a system $\mathbf{P} = (m_1, \ldots, m_n)$, of modules, where each module $m_i = (P_i[\mathbf{q_i}], R_i)$ has a module name P_i with an associated list $\mathbf{q_i}$ of formal input atoms, and an associated set of rules R_i (the "implementation"). A module m_i can access another module m_j using *module atoms* (in the body of a rule in R_i) of the form $P_j[\mathbf{p}].o$. Intuitively, the module atom evaluates to true if, on input of the atoms in $\mathbf{p}$ to the module P_j, the atom o will be true in P_j. Such programs allow unrestricted cyclic calls between modules; they can be seen as a generalization of DLP-functions from propositional to Datalog programs that allow for positive cyclic calls between modules (including recursion), and provide a call-by-value mechanism.

For example, the following MLP $\mathbf{P} = (m_1, m_2, m_3)$ recursively checks whether the number of facts over predicate q in the main module m_1, which has no input ($\mathbf{q_1}$ is empty) and implementation $R_1 = \{q(a).\ \ q(b).\ \ ok \leftarrow P_2[q].even.\}$, is even. Intuitively, m_1 calls m_2 with a rule for the check, and assigns the result to ok. The module m_2 is mutual recursive with module m_3. They have the formal inputs $\mathbf{q_2} = q_2$ and $\mathbf{q_3} = q_3$, respectively, and the implementations

$$R_2 = \left\{ \begin{array}{r} q_2'(X) \leftarrow q_2(X), q_2(Y), \\ \text{not } q_2'(Y), X \neq Y. \\ skip_2 \leftarrow q_2(X), \text{not } q_2'(X). \\ even \leftarrow \text{not } skip_2. \\ even \leftarrow skip_2, P_3[q_2'].odd. \end{array} \right\}, \quad R_3 = \left\{ \begin{array}{r} q_3'(X) \leftarrow q_3(X), q_3(Y), \\ \text{not } q_3'(Y), X \neq Y. \\ skip_3 \leftarrow q_3(X), \text{not } q_3'(X). \\ odd \leftarrow skip_3, P_2[q_3'].even. \end{array} \right\}.$$

A call to m_2 'returns' $even$, if either the input q_2 to m_2 is empty (as then $skip_2$ is false), or the call of m_3 with q_2' resulting from q_2 by randomly removing one element (then $skip_2$ is true) returns odd. Module m_3 returns odd for input q_3, if a call to m_2 with q_3' analogously constructed from q_3 returns $even$. In any answer set of $\mathbf{P}$, ok is true.

In this paper, we further the work on MLPs and turn to characterizing of answer sets in terms of classical models, in line with recent research in Answer Set Programming. To this end, we first explore the notion of *loop formulas* to MLPs. Lin and Zhao [19] first used loop formulas to characterize the answer sets of normal, i.e., disjunction-free, propositional logic programs by the models of a propositional formula built of the Clark completion [5] and of additional formulas for each positive loop in the dependency graph of the program. They built on this result developing the ASP solver ASSAT, which uses a SAT solver for answer sets computation [19]. The loop formula characterization has subsequently been extended to disjunctive logic programs [16], and to general propositional theories under a generalized notion of answer set [12]. In the latter work, the notion of a loop has been adapted to include trivial loops (singletons) in order to recast Clark's completion as loop formulas. Besides their impact on ASP solver development, loop formulas are a viable means for the study of semantic properties of ASP programs, as they allow to resort to classical logic for characterization. For instance, in the realm of modular logic programming, loop formulas have recently been fruitfully extended to DLP-functions [15], simplifying some major proofs.

The expedient properties of MLPs, however, render a generalization of loop formulas more involved. Due to the module input mechanism, it is necessary to keep track of different module instantiations. Furthermore, because of unlimited recursion in addition to loops that occur inside a module, loops across module boundaries, i.e., when modules refer to each other by module atoms, have to be captured properly. To cope with this requirements,

- we adapt Clark's completion for module atoms w.r.t. different module instantiations;
- we provide a refined version of the positive dependency graph for an MLP, the *modular dependency graph*, and *cyclic instantiation signature*: the combination then relates module instantiations with the atoms of a module;
- based on it, we define *modular loops* and their external support formulas; and
- eventually, we define *modular loop formulas*, and show that the conjunction of all modular loop formulas for an MLP characterizes the answer sets of $\mathbf{P}$ in its (Herbrand) models.

Furthermore, the definition of the MLP semantics in terms of the FLP-reduct [11] and the underlying principal idea of loop formulas requires us to restrict module atoms under negation to be monotonic. This is often not a limitation, since negated module atoms may be easily replaced by unnegated ones using a simple rewriting technique (e.g., for stratified program parts). Intuitively, the restriction seems to be the trade off for the benign property that under the FLP-reduct, answer sets of a MLP – even with nonmonotonic module atoms – are always minimal models of the program. The latter would not be the case if they were defined under the traditional GL-reduct [14], for which loop formulas have been developed.

Second, we explore the recent approach of [3] to modify the Clark completion in order to characterize answer set semantics of non-monotonic logic programs with finite Herbrand universes but without using loop formulas explicitly. The idea is to introduce predicates of the form $T_{qp}(\mathbf{y}, \mathbf{x})$ which intuitively holds when $q(\mathbf{y})$ is used to derive $p(\mathbf{x})$, and to respect a derivation order; the completion is allowed to take effect only if no positive loop is present, which is ensured by adding $T_{qp}(\mathbf{y}, \mathbf{x}) \wedge \neg T_{pq}(\mathbf{x}, \mathbf{y})$ in the completion of rules with head $p(\mathbf{x})$ and $q(\mathbf{y})$ in the positive body; for this to work, it must be ensured that T_{qp} respects transitive derivations, i.e., the composition of T_{qr} and T_{rp} must be contained in T_{qp}. The resulting translation is called *ordered completion*.

An advantage of this approach is that, at the cost of fresh (existential) predicates, constructing the (possible exponentially) many loop formulas can be avoided, while answer sets may be extracted from the (Herbrand) models of a first-order sentence, which may be fed into a suitable theorem prover. This similarly applies to MLPs, where unrestricted call-by-value however leads to an unavoidable blowup, which may be avoided by resorting to higher-order logic. Independent of computational perspectives, the novel characterizations widen our understanding of MLPs and they may prove, similarly as those in [15], useful for semantic investigations.

2 Preliminaries

We first recall syntax and semantics of modular nonmonotonic logic programs [6].

Syntax. Let $\mathcal{V}$ be a vocabulary $\mathcal{C}$, $\mathcal{P}$, $\mathcal{X}$, and $\mathcal{M}$ of mutually disjoint sets whose elements are called of *constants*, *predicate*, *variable*, and *module names*, respectively, where each $p \in \mathcal{P}$ has a fixed associated arity $n \geq 0$, and each module name in $\mathcal{M}$ has a fixed associated list $\mathbf{q} = q_1, \ldots, q_k$ ($k \geq 0$) of predicate names $q_i \in \mathcal{P}$ (the formal input parameters). Unless stated otherwise, elements from $\mathcal{X}$ (resp., $\mathcal{C} \cup \mathcal{P}$) are denoted with first letter in upper case (resp., lower case).

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. An ordinary atom (simply atom) has the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms; $n \geq 0$ is its *arity*. A *module atom* has the form $P[p_1, \ldots, p_k].o(t_1, \ldots, t_n)$, where $P \in \mathcal{M}$ is a module name with associated $\mathbf{q}$, $p_1, \ldots, p_k$ is a list of predicate names $p_i \in \mathcal{P}$, called *module input list*, such that p_i has the arity of q_i in $\mathbf{q}$, and $o \in \mathcal{P}$ is a predicate name with arity n such that for the list of terms $t_1, \ldots, t_n$, $o(t_1, \ldots, t_n)$ is an ordinary atom. Intuitively, a module atom provides a way for deciding the truth value of a ground atom $o(\mathbf{c})$ in a program P depending on the truth of a set of input atoms.

A *normal rule* r (or rule for short) is of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not}\, \beta_{m+1}, \ldots, \text{not}\, \beta_n \quad (m, n \geq 0), \tag{1}$$

where α is an atom and each β_j is an ordinary or a module atom. We define $H(r) = \{\alpha\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ and $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$. For $\star \in \{+, -\}$ we let $B^\star_m(r)$ and $B^\star_o(r)$ be the set of module and ordinary atoms that appear in $B^\star(r)$, respectively. If $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*; r is *ordinary*, if it does not contain module atoms.

We now formally define the syntax of modules and normal MLPs. A *module* is a pair $m = (P[\mathbf{q}], R)$, where $P \in \mathcal{M}$ with associated input $\mathbf{q}$, and R is a finite set of normal rules. It is either a *main module* (then $|\mathbf{q}| = 0$) or a *library module*, and is *ordinary* iff all rules in R are ordinary. We omit empty [] from (main) modules if unambiguous.

A *normal modular logic program (MLP)* is a tuple $\mathbf{P} = (m_1, \ldots, m_n)$, $n \geq 1$, where all m_i are modules and at least one is a main module, where $\mathcal{M} = \{P_1, \ldots, P_n\}$.

Example 1. Let $m_1 = (P_1[], R_1)$ with $R_1 = \{p \leftarrow P_2[p].r\}$ and $m_2 = (P_2[q], R_2)$ with $R_2 = \{r \leftarrow q\}$. Then $\mathbf{P} = (m_1, m_2)$ is a normal MLP with the main module m_1.

Example 2. Let $m_1 = (P_1[], R_1)$ with $R_1 = \{p_1 \leftarrow P_2.p_2\}$ and $m_2 = (P_2[], R_2)$ with $R_2 = \{p_2 \leftarrow P_1.p_1\}$. Putting both modules together, we get the MLP $\mathbf{P} = (m_1, m_2)$ with the main modules m_1, m_2.

W.l.o.g, in the rest of this paper, we assume that for all $i \neq j$, the atoms in m_i and m_j are distinct; thus, $\mathcal{P} = \bigcup_{i=1}^n \mathcal{P}_i$ where all $\mathcal{P}_i$ are disjoint.

Semantics. The semantics of MLPs is defined in terms of Herbrand interpretations and grounding as customary in traditional logic programming and ASP. The *Herbrand base* w.r.t. vocabulary $\mathcal{V}$, $HB_\mathcal{V}$, is the set of all ground ordinary and module atoms that can be built using $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{M}$; if $\mathcal{V}$ is implicit from an MLP $\mathbf{P}$, it is the *Herbrand base of* $\mathbf{P}$ and denoted by $HB_\mathbf{P}$. The grounding of a rule r is the set $gr(r)$ of all ground instances of r w.r.t. $\mathcal{C}$; the grounding of rule set R is $gr(R) = \bigcup_{r \in R} gr(r)$, and the one of a module m, $gr(m)$, is defined by replacing the rules in $R(m)$ by $gr(R(m))$; the grounding of an MLP $\mathbf{P}$ is $gr(\mathbf{P})$, which is formed by grounding each module m_i of $\mathbf{P}$. The semantics of an arbitrary MLP $\mathbf{P}$ is given in terms of $gr(\mathbf{P})$.

Let $S \subseteq HB_\mathbf{P}$ be any set of atoms. For any list of predicates $\mathbf{p} = p_1, \ldots, p_k$ and $\mathbf{q} = q_1, \ldots, q_k$, we use the notation $S|_\mathbf{p} = \{p_i(\mathbf{c}) \in S \mid 1 \leq i \leq k\}$ and $S|^\mathbf{q}_\mathbf{p} = \{q_i(\mathbf{c}) \mid p_i(\mathbf{c}) \in S, 1 \leq i \leq k\}$.

For a module name $P \in \mathcal{M}$ with associated formal input $\mathbf{q}$ and $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q}}$, we say that $P[S]$ is a *value call with input S*; we denote by $VC(\mathbf{P})$ the set of all such $P[S]$ for $\mathbf{P}$. Intuitively, $VC(\mathbf{P})$ names all instances of modules in $\mathbf{P}$, which we thus also use as an index set. A *rule base* is an (indexed) tuple $\mathbf{R} = (R_{P[S]} \mid P[S] \in VC(\mathbf{P}))$ of sets $R_{P[S]}$ of rules. For a module $m_i = (P_i[\mathbf{q_i}], R_i)$ from $\mathbf{P}$, its *instantiation with $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q_i}}$*, is $I_{\mathbf{P}}(P_i[S]) = R_i \cup S$. For an MLP $\mathbf{P}$, its *instantiation* is the rule base $I(\mathbf{P}) = (I_{\mathbf{P}}(P_i[S]) \mid P_i[S] \in VC(\mathbf{P}))$.

We next define (Herbrand) interpretations and models of MLPs.

Definition 1 (model). *An* interpretation $\mathbf{M}$ *of an MLP* $\mathbf{P}$ *is an (indexed) tuple* $(M_{P_i[S]} \mid P_i[S] \in VC(\mathbf{P}))$*, where all* $M_{P_i[S]} \subseteq HB_{\mathbf{P}}$ *contain only ordinary atoms. To ease notation, we also write* M_i/S *for* $M_{P_i[S]}$*. We say that* $\mathbf{M}$ *is a* model *of*

- *an atom* α *at* $P_i[S]$*, denoted* $\mathbf{M}, P_i[S] \models \alpha$*, iff (i)* $\alpha \in M_i/S$ *when* α *is ordinary, and (ii)* $o(\mathbf{c}) \in M_k/((M_i/S)|^{\mathbf{q_k}}_{\mathbf{p}})$*, when* $\alpha = P_k[\mathbf{p}].o(\mathbf{c})$ *is a module atom;*
- *a rule* r *at* $P_i[S]$ *(*$\mathbf{M}, P_i[S] \models r$*), iff* $\mathbf{M}, P_i[S] \models H(r)$ *or* $\mathbf{M}, P_i[S] \not\models B(r)$*, where (i)* $\mathbf{M}, P_i[S] \models H(r)$*, iff* $\mathbf{M}, P_i[S] \models \alpha$ *for* $H(r) = \{\alpha\}$*, and (ii)* $\mathbf{M}, P_i[S] \models B(r)$*, iff* $\mathbf{M}, P_i[S] \models \alpha$ *for all* $\alpha \in B^+(r)$ *and* $\mathbf{M}, P_i[S] \not\models \alpha$ *for all* $\alpha \in B^-(r)$*;*
- *a set of rules* R *at* $P_i[S]$ *(*$\mathbf{M}, P_i[S] \models R$*) iff* $\mathbf{M}, P_i[S] \models r$ *for all* $r \in R$*;*
- *a rule base* $\mathbf{R}$ *(*$\mathbf{M} \models \mathbf{R}$*) iff* $\mathbf{M}, P_i[S] \models R_{P_i[S]}$ *for all* $P_i[S] \in VC(\mathbf{P})$.

Finally, $\mathbf{M}$ *is a* model *of* $\mathbf{P}$*, denoted* $\mathbf{M} \models \mathbf{P}$*, iff* $\mathbf{M} \models I(\mathbf{P})$ *in case* $\mathbf{P}$ *is ground resp.* $\mathbf{M} \models gr(\mathbf{P})$*, if* $\mathbf{P}$ *is nonground. An MLP* $\mathbf{P}$ *is* satisfiable*, iff it has a model.*

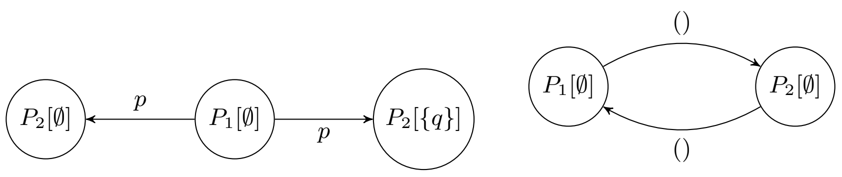
For any interpretations $\mathbf{M}$ and $\mathbf{M}'$ of $\mathbf{P}$, we define $\mathbf{M} \leq \mathbf{M}'$, iff $M_i/S \subseteq M'_i/S$ for every $P_i[S] \in VC(\mathbf{P})$, and $\mathbf{M} < \mathbf{M}'$, iff $\mathbf{M} \neq \mathbf{M}'$ and $\mathbf{M} \leq \mathbf{M}'$. A model $\mathbf{M}$ of $\mathbf{P}$ (resp., a rule base $\mathbf{R}$) is *minimal*, if $\mathbf{P}$ (resp., $\mathbf{R}$) has no model $\mathbf{M}'$ such that $\mathbf{M}' < \mathbf{M}$.

We next recall answer sets for MLPs. To focus on relevant modules, we use a *call graph*, which intuitively captures the relationship between module instances and potential module calls. The nodes correspond to module instances and edges to presumptive calls from one instance to others; edge labels distinguish different syntactical calls. Given an interpretation $\mathbf{M}$, one can determine the actual calls, starting from the main modules, following the edges whose labels match with the atoms in $\mathbf{M}$. This leads then to the *relevant call graph* with respect to $\mathbf{M}$.

Definition 2 (call graph). *The* call graph of an MLP $\mathbf{P}$ *is a labeled digraph* $CG_{\mathbf{P}} = (V, E, l)$ *with vertex set* $V = VC(\mathbf{P})$ *and an edge* e *from* $P_i[S]$ *to* $P_k[T]$ *in* E *iff* $P_k[\mathbf{p}].o(\mathbf{t})$ *occurs in* R_i*, and* e *has the input list* $\mathbf{p}$ *in its label, i.e.,* $\mathbf{p} \in l(e)$*. Given an interpretation* $\mathbf{M}$ *of* $\mathbf{P}$*, the* relevant call graph $CG_{\mathbf{P}}(\mathbf{M}) = (V', E')$ of $\mathbf{P}$ w.r.t. $\mathbf{M}$ *is the smallest subgraph of* $CG_{\mathbf{P}}$ *such that* E' *contains all edges from* $P_i[S]$ *to* $P_k[T]$ *of* $CG_{\mathbf{P}}$ *where* $(M_i/S)|^{\mathbf{q_k}}_{l(e)} = T$ *and* V' *contains all* $P_i[S]$ *that are main module instantiations or induced by* E'*; any such* $P_i[S]$ *is called* relevant w.r.t. $\mathbf{M}$.

For instance, the call graphs of the MLPs in Example 1 and 2 are shown in Fig. 1a and 1b, respectively.

For answer sets of an MLP $\mathbf{P}$, we use a reduct of the instantiated program as customary in ASP; for reasons discussed in [6], we use the FLP reduct [11] rather than the traditional Gelfond-Lifschitz reduct [14]. As $\mathbf{P}$ might have inconsistent module instantiations, which compromises the existence of an answer set of $\mathbf{P}$, we contextualize



(a) Example 1 (b) Example 2

Fig. 1: Call graphs

reducts and answer sets. Let $V(G)$ and $E(G)$ denote the vertex and edge set of a graph G, respectively.

Definition 3 (context-based reduct). *A* context *for an interpretation* $\mathbf{M}$ *of an MLP* $\mathbf{P}$ *is any set* $C \subseteq VC(\mathbf{P})$ *such that* $V(CG_{\mathbf{P}}(\mathbf{M})) \subseteq C$*. The* reduct of $\mathbf{P}$ at $P[S]$ w.r.t. $\mathbf{M}$ and C*, denoted* $f\mathbf{P}(P[S])^{\mathbf{M},C}$*, is the rule set* $I_{gr(\mathbf{P})}(P[S])$ *from which, if* $P[S] \in C$*, all rules* r *such that* $\mathbf{M}, P[S] \not\models B(r)$ *are removed. The* reduct of $\mathbf{P}$ w.r.t. $\mathbf{M}$ and C *is* $f\mathbf{P}^{\mathbf{M},C} = (f\mathbf{P}(P[S])^{\mathbf{M},C} \mid P[S] \in VC(\mathbf{P}))$*.*

That is, outside C the module instantiations of $\mathbf{P}$ remain untouched, while inside C the FLP-reduct [11] is applied.

Definition 4 (answer set). *Let* $\mathbf{M}$ *be an interpretation of a ground MLP* $\mathbf{P}$*. Then* $\mathbf{M}$ *is an* answer set of $\mathbf{P}$ w.r.t. a context C for $\mathbf{M}$*, iff* $\mathbf{M}$ *is a minimal model of* $f\mathbf{P}^{\mathbf{M},C}$*.*

In particular, if $\mathbf{P} = (m_1)$ consists of a single module m_1 with no calls to itself, the answer sets of $\mathbf{P}$ coincide with the answer sets of R_1.

Note that C is a parameter that allows to select a degree of overall-stability for answer sets of $\mathbf{P}$. The minimal context $C = V(CG_{\mathbf{P}}(\mathbf{M}))$ is the relevant call graph of $\mathbf{P}$. For the rest of this paper, we assume that $C = VC(\mathbf{P})$, i.e., all module instances have answer sets (see Section 7 for further discussion).

Example 3. The program in Example 1 has the single answer set $(M_1/\emptyset := \emptyset, M_2/\emptyset := \emptyset, M_2/\{q\} := \{r, q\})$ while the program in Example 2 has the single answer set $(M_1/\emptyset := \emptyset, M_2/\emptyset := \emptyset)$.

A module atom $\beta = P_k[\mathbf{p}].o(\mathbf{c})$ that appears in a rule in $m_i = (P_i[\mathbf{q_i}], R_i)$ in an MLP $\mathbf{P}$ is *monotonic*, if for all interpretations $\mathbf{M}, \mathbf{N}$ of $\mathbf{P}$ such that $\mathbf{M} \leq \mathbf{N}$ and $\mathbf{M} \neq \mathbf{N}$, and all $P_i[S] \in VC(\mathbf{P})$, we have that $\mathbf{M}, P_i[S] \models \beta$ implies $\mathbf{N}, P_i[S] \models \beta$.

In the sequel, we will characterize the answer sets of MLPs via loop formulas and program completion where all module atoms under negation are monotonic. Such characterizations consist of two parts: (1) the completion, which singles out classical models, which is studied in Section 3; (2) the loop formulas, which take care of minimality (foundedness) aspects; this will be considered in Section 4. Alternatively, the completion can be made ordered, which we do in Section 5.

3 Program Completion for MLPs

We start with adapting the classical Clark completion [5] with module atoms. The intuition behind this adaption is to replace every module atom $\beta = P_k[\mathbf{p}].o(\mathbf{c})$ in m_i

by a formula $\mu(m_i, \beta, S)$ which selects, based on the value of the input atoms $\mathbf{p}$ in the value call $P_i[S]$, the "right" instance $P_k[T]$ of P_k and retrieves the value of $o(\mathbf{c})$ in it.

Given a set $S \subseteq HB_{\mathbf{P}}$ of ordinary atoms, we assume that S is enumerated, i.e., $S = \{a_1, \ldots, a_n\}$ where $n = |S|$. We identify subsets B of S with their characteristic function $\chi^B : S \to \{0, 1\}$ such that $\chi^B(a) = 1$ iff $a \in B$.

For any ordinary atom $a \in HB_{\mathbf{P}}$ and any set of ordinary atoms A, let a^A denote a fresh atom, and for any set $B \subseteq HB_{\mathbf{P}}$ of ordinary atoms, let $B^A = \{a^A \mid a \in B\}$. Next, we define support rules. Intuitively, support rules are used to define the completion of an atom. The *support rules* of a set of rules R w.r.t. an ordinary atom $\alpha \in HB_{\mathbf{P}}$ is

$$SR(\alpha, R) = \{r \in R \mid H(r) = \{\alpha\}\} \ .$$

Let $\neg.A = \{\neg a \mid a \in A\}$ and, as usual, $\bigvee F = \bigvee_{f \in F} f$ and $\bigwedge F = \bigwedge_{f \in F} f$ (note that $\bigvee \emptyset = \bot$ and $\bigwedge \emptyset = \top$).

Then, for every module atom $\beta = P_k[\mathbf{p}].o(\mathbf{c}) \in HB_{\mathbf{P}}$ from some module $m_i = (P_i[\mathbf{q_i}], R_i)$ (where P_k has formal input $\mathbf{q_k} = q_{k,1}, \ldots, q_{k,n_k}$) and $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q_i}}$, let

$$\beta_{m_i,S,T} = \bigwedge_{\chi^T(q_{k,j}(\mathbf{c}))=1} p_j^S(\mathbf{c}) \wedge \bigwedge_{\chi^T(q_{k,j}(\mathbf{c}))=0} \neg p_j^S(\mathbf{c})$$

and

$$\mu(m_i, \beta, S) = \bigvee_{T \subseteq HB_{\mathbf{P}}|_{\mathbf{q_k}}} \left(\beta_{m_i,S,T} \wedge o^T(\mathbf{c})\right) \ ,$$

$$\bar{\mu}(m_i, \beta, S) = \bigvee_{T \subseteq HB_{\mathbf{P}}|_{\mathbf{q_k}}} \left(\beta_{m_i,S,T} \wedge \neg o^T(\mathbf{c})\right) \ .$$

We can now define the modular completion, which relates instantiations of the rules in modules to propositional formulas.

Definition 5 (Modular Completion). *Let r be a rule from module $m_i = (P_i[\mathbf{q_i}], R_i)$, and let $S \subseteq HB_{\mathbf{P}}|_{\mathbf{q_i}}$. Then*

$$\begin{aligned}\gamma(m_i, r, S) = & \bigwedge B_o^+(r)^S \wedge \bigwedge_{\beta \in B_m^+(r)} \mu(m_i, \beta, S) \wedge \\ & \bigwedge \neg.B_o^-(r)^S \wedge \bigwedge_{\beta \in B_m^-(r)} \bar{\mu}(m_i, \beta, S) \supset H(r)^S \ ,\end{aligned} \tag{2}$$

and

$$\begin{aligned}\sigma(m_i, r, S) = & \bigwedge B_o^+(r)^S \wedge \bigwedge_{\beta \in B_m^+(r)} \mu(m_i, \beta, S) \wedge \\ & \bigwedge \neg.B_o^-(r)^S \wedge \bigwedge_{\beta \in B_m^-(r)} \bar{\mu}(m_i, \beta, S) \ ,\end{aligned} \tag{3}$$

For a set of rules R, we let $\sigma(m_i, R, S) = \bigvee_{r \in R} \sigma(m_i, r, S)$ and $\gamma(m_i, R, S) = \bigwedge_{r \in R} \gamma(m_i, r, S)$.

For any value call $P_i[S]$ of module $m_i = (P_i[\mathbf{q_i}], R_i)$, $\mathbf{q_i} = q_{i,1}, \ldots, q_{i,n_i}$, in $\mathbf{P}$, let

$$\gamma(\mathbf{P}, P_i[S]) = \gamma(m_i, R_i, S) \wedge \bigwedge_{\chi^S(q_{i,j}(\mathbf{c}))=1} q^S_{i,j}(\mathbf{c}) \ , \tag{4}$$

$$\sigma(\mathbf{P}, P_i[S]) = \bigwedge_{r \in R_i, a \in H(r)} a^S \supset \sigma(m_i, SR(a, R_i), S) \tag{5}$$

and

$$\gamma(\mathbf{P}) = \bigwedge_{P_i[S] \in VC(\mathbf{P})} \gamma(\mathbf{P}, P_i[S]) \ ,$$

$$\sigma(\mathbf{P}) = \bigwedge_{P_i[S] \in VC(\mathbf{P})} \sigma(\mathbf{P}, P_i[S]) \ .$$

Example 4. Continuing with $\mathbf{P}$ of Example 1, we get the following formulas (here, $S_1 = \emptyset$, $S_2^0 = \emptyset$ and $S_2^1 = \{q\}$):
- $\gamma(\mathbf{P}, P_1[\emptyset]) = (\neg p^{S_1} \wedge r^{S_2^0}) \vee (p^{S_1} \wedge r^{S_2^1}) \supset p^{S_1}$,
- $\gamma(\mathbf{P}, P_2[\emptyset]) = q^{S_2^0} \supset r^{S_2^0}$,
- $\gamma(\mathbf{P}, P_2[\{q\}]) = \left(q^{S_2^1} \supset r^{S_2^1}\right) \wedge q^{S_2^1}$,
- $\sigma(\mathbf{P}, P_1[\emptyset]) = p^{S_1} \supset (\neg p^{S_1} \wedge r^{S_2^0}) \vee (p^{S_1} \wedge r^{S_2^1})$,
- $\sigma(\mathbf{P}, P_2[\emptyset]) = r^{S_2^0} \supset q^{S_2^0}$,
- $\sigma(\mathbf{P}, P_2[\{q\}]) = r^{S_2^1} \supset q^{S_2^1}$.

The conjunction of the first three formulas yields $\gamma(\mathbf{P})$, and the last three give us $\sigma(\mathbf{P})$.

Example 5. For the MLP $\mathbf{P}$ in Example 2, we get the following formulas ($S = \emptyset$):
- $\gamma(\mathbf{P}) = p_2^S \supset p_1^S \wedge p_1^S \supset p_2^S$
- $\sigma(\mathbf{P}) = p_1^S \supset p_2^S \wedge p_2^S \supset p_1^S$

The formula $\gamma(\mathbf{P})$ now captures the (classical) models of $\mathbf{P}$.

Lemma 1. *The models of $\gamma(\mathbf{P})$ correspond 1-1 to the models of $\mathbf{P}$. That is, (i) if $M \models \gamma(\mathbf{P})$, then $\mathbf{M} \models \mathbf{P}$, where $M_i/S = \{p(\mathbf{c}) \in HB_\mathbf{P} \mid p^S(\mathbf{c}) \in M \wedge p \in \mathcal{P}_i\}$, for all $P_i[S]$, and (ii) if $\mathbf{M} \models \mathbf{P}$, then $M \models \gamma(\mathbf{P})$, where $M = \bigcup_{P_i[S]}(M_i/S)^S$.*

Proof (sketch). (i) Suppose $M \models \gamma(\mathbf{P})$, and let $\mathbf{M}$ as described. We need to show that $\mathbf{M}, P_i[S] \models r$ for each $r \in I_\mathbf{P}(P_i[S]) = R_i \cup S$ and $P_i[S] \in VC(\mathbf{P})$. If r is a fact $q_j(\mathbf{c})$ for a formal input parameter q_j of $P_i[\mathbf{q}]$, then $q_j(\mathbf{c}) \in S$ and, by formula (4), $M \models q_j^S(\mathbf{c})$; hence, $q_j(\mathbf{c}) \in M_i/S$, and thus $\mathbf{M}, P_i[S] \models r$. Otherwise, $r \in R_i$. As $M \models \gamma(m_i, R_i, S)$, we have that M satisfies the formula (2). By construction, for each ordinary atom β in r, we have $M \models \beta^S$ iff $\mathbf{M}, P_i[S] \models \beta$; furthermore, $M \models \mu(m_i, \beta, S)$ for $P_k[\mathbf{p}].o(\mathbf{c})$ iff $M \models o^T(\mathbf{c})$, where $T \subseteq HB_\mathbf{P}|_{\mathbf{q_k}}$ is the unique set T such that $M \models \bigwedge_j (p_j^S(\mathbf{c}) \equiv q_{i,j}^T(\mathbf{c}))$. That is, $M \models \mu(m_i, \beta, S)$ iff $\mathbf{M}, P_i[S] \models \beta$. Hence, it follows that $\mathbf{M}, P_i[S] \models r$.
(ii) Suppose $\mathbf{M} \models \mathbf{P}$, and let $M = \bigcup_{P_i[S]}(M_i/S)^S$. To show that $M \models \gamma(\mathbf{P})$, we must show that $M \models \gamma(\mathbf{P}, P_i[S])$ for all $P_i[S]$. As $S \subseteq I_\mathbf{P}(P_i[S])$ and $\mathbf{M}, P_i[S] \models$

$I_{\mathbf{P}}(P_i[S])$, all conjuncts q_j^S (representing the formal input) in $\gamma(\mathbf{P}, P_i[S])$ are satisfied by M; thus it remains to show $M \models \gamma(m_i, R_i, S)$, i.e., $M \models \gamma(m_i, r, S)$ for each $r \in R_i$. For each ordinary atom β in r, we have by construction of M that $M \models \beta^S$ iff $\mathbf{M}, P_i[S] \models \beta$; furthermore, for each module atom $\beta = P_k[\mathbf{p}].o(\mathbf{c})$ in r, we have that $M \models \mu(m_i, \beta, S)$ iff $M \models o^T(\mathbf{c})$, i.e., $o(\mathbf{c}) \in M_k/T$, where $T \subseteq HB_{\mathbf{P}}|_{\mathbf{q_k}}$ contains $q_{k,j}(\mathbf{c})$ iff $M \models p_j^S(\mathbf{c})$, i.e., $p_j(\mathbf{c}) \in M_i/S$. Thus, $M \models \mu(m_i, \beta, S)$ iff $o(\mathbf{c}) \in M_k/(M_i/S)|_{\mathbf{p}}^{\mathbf{q_k}}$; in other words, iff $\mathbf{M}, P_i[S] \models o(\mathbf{c})$. As $\mathbf{M}, P_i[S] \models r$, it follows that $M \models \gamma(m_i, r, S)$.

Call a model $\mathbf{M}$ of $\mathbf{P}$ *supported*, if for every atom $\alpha \in M_i/S$, $P_i[S] \in VC(\mathbf{P})$, there is some rule $r \in SR(\alpha, I_{\mathbf{P}}(P_i[S]))$ such that $\mathbf{M}, P_i[S] \models B(r)$. Then, based on Lemma 1 the following can be shown.

Lemma 2. *The models of* $\gamma(\mathbf{P}) \wedge \sigma(\mathbf{P})$ *correspond 1-1 to the supported models of* $\mathbf{P}$.

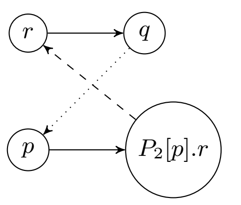
In particular, if $\mathbf{P}$ is acyclic (no atom depends recursively on itself), then it has a single supported model, which gives rise to an answer set of $\mathbf{P}$.

Example 6. Continuing with Example 4, we get for $\gamma(\mathbf{P})$ the classical models $M_1 = \{r^{S_2^1}, q^{S_2^1}\}$, $M_2 = \{p^{S_1}, r^{S_2^0}, r^{S_2^1}, q^{S_2^1}\}$, $M_3 = \{p^{S_1}, r^{S_2^1}, q^{S_2^1}\}$, and $M_4 = \{p^{S_1}, r^{S_2^0}, r^{S_2^1}, q^{S_2^0}, q^{S_2^1}\}$. They correspond to the classical models $\mathbf{M}_1 = (M_1/\emptyset := \emptyset, M_2/\emptyset := \emptyset, M_2/\{q\} := \{r, q\})$, $\mathbf{M}_2 = (M_1/\emptyset := \{p\}, M_2/\emptyset := \{r\}, M_2/\{q\} := \{r, q\})$, $\mathbf{M}_3 = (M_1/\emptyset := \{p\}, M_2/\emptyset := \emptyset, M_2/\{q\} := \{r, q\})$, and $\mathbf{M}_4 = (M_1/\emptyset := \{p\}, M_2/\emptyset := \{r, q\}, M_2/\{q\} := \{r, q\})$ of $\mathbf{P}$. The formula $\gamma(\mathbf{P}) \wedge \sigma(\mathbf{P})$ has only the classical models M_1,M_3, and M_4, which will give us the supported models $\mathbf{M}_1$, $\mathbf{M}_2$, and $\mathbf{M}_4$ of $\mathbf{P}$.

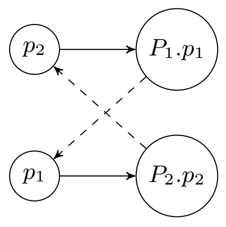
Example 7. In Example 5, the models of $\gamma(\mathbf{P})$ are $M_1 = \emptyset$ and $M_2 = \{p_1^S, p_2^S\}$, which are also the models of $\gamma(\mathbf{P}) \wedge \sigma(\mathbf{P})$. Both of them correspond to the classical as well as supported models of $\mathbf{P}$, namely $\mathbf{M}_1 = (M_1/\emptyset := \emptyset, M_2/\emptyset := \emptyset)$ and $\mathbf{M}_2 = (M_1/\emptyset := \{p_1\}, M_2/\emptyset := \{p_2\})$.

4 Loop Formulas for MLPs

In this section, we develop *modular loop formulas* that instantiate each program module with possible input to create the classical theory of the program, and then add loop formulas similar as in [16]. However, we have to respect loops not only inside a module, but also across modules due to module atoms. The latter will be captured by a *modular dependency graph*, which records positive dependencies that relates module instantiations with the atoms in a module. The instantiation of the modules makes it necessary to create fresh propositional atoms very similar to grounding of logic programs; complexity results in [6] suggest that there is no way to circumvent this: with arbitrary input, already propositional Horn MLPs are EXP-complete and normal propositional MLPs are NEXP-complete (considering brave inference of a ground atom, i.e., membership in some answer set). In the non-ground case, Horn MLPs are 2EXP-complete, while normal non-ground MLPs are 2NEXP-complete. In the rest of this section, we assume that all negated module atoms in a MLP are monotonic and that $\mathbf{P}$ is ground.



(a) Example 1



(b) Example 2

Fig. 2: Module dependency graphs

We define now the modular dependency graph to keep track of dependencies between modules and rules. It is a ground dependency graph with two additional types of edges.

Definition 6 (Modular Dependency Graph). *Let* $\mathbf{P} = (m_1, \ldots, m_n)$ *be an MLP. The dependency graph of* $\mathbf{P}$ *is the digraph* $MG_{\mathbf{P}} = (V, E)$ *with vertex set* $V = HB_{\mathbf{P}}$ *and edge set* E *containing the following edges:*

– $p(\mathbf{c_1}) \rightarrow q(\mathbf{c_2})$, *for each* $r \in R_i$ *with* $H(r) = \{p(\mathbf{c_1})\}$ *and* $q(\mathbf{c_2}) \in B^+(r)$.
– $a \rightarrow b$, *if one of* (i)–(ii) *holds, where* α *is of the form* $P_j[\mathbf{p}].o(\mathbf{c})$ *in* R_i *and* P_j *has the associated input list* $\mathbf{q_j}$*:*
  (i) $a = \alpha$ *and* $b = o(\mathbf{c}) \in HB_{\mathbf{P}}$*;*
  (ii) $a = q_\ell(\mathbf{c}) \in HB_{\mathbf{P}}|_{\mathbf{q_j}}$ *and* $b = p_\ell(\mathbf{c}) \in HB_{\mathbf{P}}|_{\mathbf{p}}$ *for* $1 \leq \ell \leq |\mathbf{q_j}|$.

Intuitively, the module graph is “uninstantiated”, i.e., all module atoms are purely syntactic. This also means that loops that show up in the module graph must be “instantiated” in the formulas.

Example 8. The module dependency graphs of the programs in Examples 1 and 2 are shown in Fig. 2a and 2b, resp. In both figures, the two upper nodes are from m_2, while the nodes below stem from m_1. Note that the dashed edges stem from condition (i) in Definition 6, while dotted edges are from condition (ii). Straight edges are standard head-body dependencies.

We define now modular loops based on the modular dependency graph.

Definition 7 (Modular Loops). *A set of atoms* $\mathcal{L} \subseteq V(MG_{\mathbf{P}})$ *is called a* modular loop for $\mathbf{P}$ *iff the subgraph of* $MG_{\mathbf{P}}$ *induced by* $\mathcal{L}$ *is strongly connected.*

Note that $\mathcal{L}$ may contain module atoms, and single-atom loops are allowed.

Modular loop formulas have then the same shape as standard loop formulas [19, 16], with the important distinction that external support formulas may take the input S from the value call $P_i[S]$. For that, we define first the *external support rules* of rule set R w.r.t. a set $\mathcal{L} \subseteq HB_{\mathbf{P}}$ as

$$ER(\mathcal{L}, R) = \{r \in R \mid H(r) \cap \mathcal{L} \neq \emptyset,\ B^+(r) \cap \mathcal{L} = \emptyset\} \ .$$

Note that $\mathcal{L}$ may contain module atoms.

Modular loops may go through the atoms of multiple modules, but do not take care of “instantiated loops” that stem from the input. Given a modular loop, the instantiated loop may be exponentially longer in the propositional case, whereas it could have double-exponential length in the non-ground case. To keep record of these loops, we next define cyclic instantiation signatures that are used to instantiate modular loops.

Definition 8 (Cyclic instantiation signature). *Let $\mathcal{L}$ be a modular loop for $\mathbf{P} = (m_1, \ldots, m_n)$. A* cyclic instantiation signature for $\mathcal{L}$ *is a tuple $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)$ such that for all $i \in \{1, \ldots, n\}$, (i) $\mathcal{S}_i \subseteq 2^{HB_{\mathbf{P}}|_{\mathbf{q_i}}}$ with $\mathcal{S}_i \neq \emptyset$ and all $S \in \mathcal{S}_i$ have $S \cap \mathcal{L} = \emptyset$, if $\mathcal{L}$ has some ground atoms with predicates from $\mathcal{P}_i$, and (ii) $\mathcal{S}_i = \emptyset$ otherwise.*

Intuitively, we use a modular loop as template to create loops that go over instantiations.

Example 9. The MLP $\mathbf{P}$ in Example 1 has the loop $\mathcal{L} = \{p, P_2[p].r, r, q\}$ for which we get one cyclic instantiation signature $\mathcal{S}_1 = (\{\emptyset\}, \{\emptyset\})$; $(\{\emptyset\}, \{\{q\}\})$ and $(\{\emptyset\}, \{\emptyset, \{q\}\})$ are not cyclic instantiation signatures as they share atoms with $\mathcal{L}$, thus always get support from input S. Intuitively, this captures those module instantiations that cycle over module input, but have no support from the formal input, viz., $P_1[\emptyset] \leftrightarrow P_2[\emptyset]$.

Example 10. In Example 2, we have a loop $\mathcal{L} = \{p_1, P_2.p_2, p_2, P_1.p_1\}$. We get one cyclic instantiation signatures: $\mathcal{S}_1 = (\{\emptyset\}, \{\emptyset\})$. Here, $\mathcal{S}_1$ builds a cycle over module instantiations from the mutual calls in m_1 and m_2.

Definition 9 (Modular Loop Formulas). *Let $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)$ be an instantiation signature for the modular loop $\mathcal{L}$ in MLP $\mathbf{P}$. The* loop formula for $\mathcal{L}$ w.r.t. $\mathcal{S}$ in $\mathbf{P}$ *is*

$$\lambda(\mathcal{S}, \mathcal{L}, \mathbf{P}) \quad = \quad \bigvee_{i=1}^{n} \bigvee_{T \in \mathcal{S}_i} \left(\bigvee (\mathcal{L}|_{\mathcal{P}_i})^T \right) \quad \supset \quad \bigvee_{i=1}^{n} \bigvee_{S \in \mathcal{S}_i} \sigma(m_i, ER(\mathcal{L}, R_i), S) \enspace .$$

Given $\mathbf{P}$*, the* loop formula for a modular loop $\mathcal{L}$ in $\mathbf{P}$ *is the conjunction $\lambda(\mathcal{L}, \mathbf{P}) = \bigwedge_{\mathcal{S}} \lambda(\mathcal{S}, \mathcal{L}, \mathbf{P})$ for all cyclic instantiation signatures $\mathcal{S}$ of $\mathcal{L}$, and the* loop formula for $\mathbf{P}$ *is the conjunction $\lambda(\mathbf{P}) = \bigwedge_{\mathcal{L}} \lambda(\mathcal{L}, \mathbf{P})$ for all modular loops $\mathcal{L}$ in $\mathbf{P}$.*

Intuitively, the formal input in a value call $P_i[S]$ always adds external support for the input atoms in S as we add S to the instantiation $I_{\mathbf{P}}(P_i[S])$. Since we obtain all supported models with $\gamma(\mathbf{P}) \wedge \sigma(\mathbf{P})$, thus also have S there, we can restrict to those instantiation signatures $\mathcal{S}$ for a modular loop $\mathcal{L}$ that have no support from formal input. Putting things together, let us define

$$\Lambda(\mathbf{P}) = \gamma(\mathbf{P}) \wedge \sigma(\mathbf{P}) \wedge \lambda(\mathbf{P}) \enspace .$$

Example 11. Continuing with Example 4, we get the following modular loop formulas based on the loop $\mathcal{L}$ and instantiation signature $\mathcal{S}_1$ for $\mathcal{L}$ shown in Example 9 (here, $S_1 = \emptyset$, $S_2^0 = \emptyset$): $\lambda(\mathcal{S}_1, \mathcal{L}, \mathbf{P}) = (p^{S_1} \vee r^{S_2^0} \vee q^{S_2^0}) \supset \bot$. This formula and $\gamma(\mathbf{P}) \wedge \sigma(\mathbf{P})$ yields $\Lambda(\mathbf{P})$, whose model is $M_1 = \{r^{S_2^1}, q^{S_2^1}\}$, which coincides with the answer set $\mathbf{M}_1 = (\emptyset, \emptyset, \{r, q\})$ of $\mathbf{P}$.

Example 12. Based on Example 5 and 10 we get the following modular loop formulas using the loop $\mathcal{L}$ and instantiation signature $\mathcal{S}_1$ ($S = \emptyset$): $\lambda(\mathcal{S}_1, \mathcal{L}, \mathbf{P}) = p_1^S \vee p_2^S \supset \bot \vee \bot$. The classical model of the conjunction of $\gamma(\mathbf{P}) \wedge \sigma(\mathbf{P})$ and above formula ($= \Lambda(\mathbf{P})$) is thus $M_1 = \emptyset$, which coincides with the answer set $\mathbf{M}_1 = (\emptyset, \emptyset)$ of $\mathbf{P}$.

We have the following result.

Theorem 1. *Given an MLP $\mathbf{P}$ in which all negated module atoms are monotonic, the answer sets of $\mathbf{P}$ and the classical models of $\Lambda(\mathbf{P})$ correspond, such that (i) if $M \models \Lambda(P)$, then there is some answer set $\mathbf{M}$ of $\mathbf{P}$ such that $M_i/S = \{p(\mathbf{c}) \in HB_{\mathbf{P}} \mid p^S(\mathbf{c}) \in M \wedge p \in \mathcal{P}_i\}$ for all $P_i[S] \in VC(\mathbf{P})$, and (ii) if $\mathbf{M}$ is an answer set of $\mathbf{P}$, then $M \models \Lambda(\mathbf{P})$, where $M = \bigcup_{P_i[S]}(M_i/S)^S$.*

5 Ordered Completion for MLPs

In this section, we follow the idea in [3] to provide an ordered completion for non-ground MLPs. We consider MLPs in the Datalog setting, i.e., an MLP $\mathbf{P}$ can be viewed as a *modular nonmonotonic Datalog program* which has an infinite set of constants $\mathcal{C}$ and is independent from the domains (this is ensured by forcing safety conditions to rules in $\mathbf{P}$). Grounding of $\mathbf{P}$ is done with respect to a *finite relational structure* $\mathfrak{M}$ (extended to MLPs), having a finite universe $U_{\mathfrak{M}}$ accessible by constants; it is the active domain we are restricted to. We also need to adapt the notion of answer set for this setting. Like above, we assume that all negated module atoms in a MLP are monotonic. Moreover, we assume that MLPs do not contain facts, i.e., rules of form (1) have non-empty body.[1]

Finite Structures for MLPs. Given an MLP $\mathbf{P}$, we call a predicate in $\mathbf{P}$ *intensional* if it occurs in the head of a rule in $\mathbf{P}$ or in the formal input parameters $\mathbf{q_i}$ of a module $m_i = (P_i[\mathbf{q_i}], R_i)$, and *extensional* otherwise. Intuitively, intensional predicates are defined by the rules in $\mathbf{P}$ and the input given to a module instantiation, whereas extensional predicates stem from the extension given by a relational structure. The *signature* of an MLP $\mathbf{P}$ contains all intensional predicates, extensional predicates, and constants occurring in $\mathbf{P}$. The set of intensional (resp., extensional) predicates in a module m is denoted by $\mathit{Int}(m)$ (resp., $\mathit{Ext}(m)$).

A finite (Herbrand) relational structure for $\mathbf{P}$ (H-structure) can be defined as a pair $\mathfrak{M} = (U_{\mathfrak{M}}, \cdot^{\mathfrak{M}})$, where the finite universe $U_{\mathfrak{M}}$ consists of constants in $\mathbf{P}$ and $\cdot^{\mathfrak{M}}$ is a mapping associating (i) each constant in $\mathbf{P}$ with itself, i.e., $c^{\mathfrak{M}} = c$, (ii) each extensional predicate q in $\mathbf{P}$ with a relation $q^{\mathfrak{M}}$ over $\mathfrak{M}$, where $q^{\mathfrak{M}}$ has the same arity as q, (iii) each intensional predicate p in a module $m_i = (P_i[\mathbf{q}_i], R_i)$, together with each input S from the value calls $P_i[S] \in VC(\mathbf{P})$, with a relation $p^{\mathfrak{M},S}$ whose arity is the same as p.

The grounding process is gradually defined as follows. The grounding of a rule r under $\mathfrak{M}$ is the set $gr(r, \mathfrak{M})$ of all ground instances of r by replacing all variables in the rules by some domain objects in $\mathfrak{M}$. The grounding of a rule set R w.r.t. $\mathfrak{M}$ is $gr(R, \mathfrak{M}) = \bigcup_{r \in R} gr(r, \mathfrak{M}) \cup \{q(\mathbf{c}) \mid q \in \mathit{Ext}(R) \wedge \mathbf{c} \in q^{\mathfrak{M}}\}$; intuitively, it means that rules are grounded wrt. $\mathfrak{M}$ and facts are taken from the finite structure as a database. The grounding of a module m wrt. $\mathfrak{M}$, denoted by $gr(m, \mathfrak{M})$, is defined by replacing the rules in $R(m)$ by $gr(R(m), \mathfrak{M})$. Finally, the grounding of $\mathbf{P}$ wrt. $\mathfrak{M}$ is $gr(\mathbf{P}, \mathfrak{M})$, which is formed by grounding each module m_i of $\mathbf{P}$ wrt. $\mathfrak{M}$.

[1] This is w.l.o.g., since we can remove facts from an MLP and map them to extensional relations in a finite relational structure.

We say that $\mathfrak{M}$ is an answer set of $\mathbf{P}$ iff the interpretation $\mathbf{M} = (M_i/S \mid P_i[S] \in \mathit{VC}(\mathbf{P}))$, where

$$M_i/S = \{q(\mathbf{c}) \mid q \in \mathit{Ext}(m_i) \wedge \mathbf{c} \in q^{\mathfrak{M}}\} \cup \{p(\mathbf{c}) \mid p \in \mathit{Int}(m_i) \wedge \mathbf{c} \in p^{\mathfrak{M},S}\} \ ,$$

is an answer set of $gr(\mathbf{P}, \mathfrak{M})$ according to Definition 4.

Ordered Completion. Given an MLP $\mathbf{P}$, our goal now is to give a translation of $\mathbf{P}$ to a first-order formula such that the models of the latter correspond to the answer sets of the former.

Suppose that we are in a value call $P_i[S]$ of the module $m_i = (P_i[\mathbf{q}_i], R_i)$ and consider a module atom $\beta = P_j[\mathbf{p}].o(\mathbf{y})$ from a module $m_j = (P_j[\mathbf{q}_j], R_j)$. Formula $\beta_{m_i,S,T}$, as defined in Section 3, can then match the interpretation of $\mathbf{p}$ to an input T of β.

Once this is done, we can "guess" the right T by ranging over all possible subsets of the restricted Herbrand base $HB_{\mathbf{P}}|_{\mathbf{q_j}}$ of the called module m_j with formal input parameters $\mathbf{q_j}$. For each guess, we translate the module atom β to its output predicate labeled with the corresponding input T. Depending on whether β appears in the positive (resp. negative) part of the body of a rule, one uses the translation μ' (resp., $\overline{\mu'}$):

$$\begin{aligned}
\mu'(m_i, \beta, S) &= \bigvee_{T \subseteq HB_{\mathbf{P}}|_{\mathbf{q}_j}} \beta_{m_i,S,T} \wedge o^T(\mathbf{y}) \\
\overline{\mu'}(m_i, \beta, S) &= \bigvee_{T \subseteq HB_{\mathbf{P}}|_{\mathbf{q}_j}} \beta_{m_i,S,T} \wedge \neg o^T(\mathbf{y}) \ .
\end{aligned}$$

Compared to μ and $\overline{\mu}$, the primed version here deals with non-ground output of β.

Now we can build the left direction of the completion, which intuitively says that if there is some ground body which holds, then the respective head is concluded. We assume that rules are standardized apart, i.e., a predicate a appearing in the head of a rule always has the form $a(\mathbf{x})$. Suppose that free variables in the body of a rule r are $\mathbf{y}_1, \ldots, \mathbf{y}_n$, the left hand side of the completion $\gamma'(m_i, a(\mathbf{x}), S)$ is a lifted version of $\gamma(m_i, r, S)$ to the non-ground case which merges all supporting rules for $a(\mathbf{x})$. We define

$$\begin{aligned}
&\gamma'(m_i, a(\mathbf{x}), S) = \\
&\quad \forall \mathbf{x} \left[\bigvee_{r \in SR(a(\mathbf{x}), R_i)} \exists \mathbf{y}_1, \ldots, \mathbf{y}_n \left(\bigwedge_{p_l(\mathbf{y}_l) \in B_o^+(r)} p_l^S(\mathbf{y}_l) \wedge \bigwedge_{p_l(\mathbf{y}_l) \in B_o^-(r)} \neg p_l^S(\mathbf{y}_l) \wedge \right. \right. \\
&\qquad \left. \left. \bigwedge_{\beta \in B_m^+(r)} \mu'(m_i, \beta, S) \wedge \bigwedge_{\beta \in B_m^-(r)} \overline{\mu'}(m_i, \beta, S) \right) \supset a^S(\mathbf{x}) \right] \ .
\end{aligned}$$

Example 13. Take $\mathbf{P}$ from Example 1 and the labels S_1, S_2^0, and S_2^1 from Example 11. We have $\gamma'(m_1, p, \emptyset) = (\neg p^{S_1} \wedge r^{S_2^0}) \vee (p^{S_1} \wedge r^{S_2^1}) \supset p^{S_1}$.

We next turn to build the right hand side of the completion, with a modification concerning the order between predicates. Similar to [3], we use a predicate D to keep track of the derivation/dependency ordering between labeled predicates. Basically, D is labeled with subscripts describing the two related predicates (the former is used in deriving the latter, in a transitive way), and superscripts referring to the respective inputs. For example, $D^{TS}_{oa}(\mathbf{y},\mathbf{x})$ means that $o(\mathbf{y})$ in a value call $P_j[T]$ is used to derive $a(\mathbf{x})$ in $P_i[S]$; hence, $D^{TS}_{oa}(\mathbf{y},\mathbf{x}) \wedge \neg D^{ST}_{ao}(\mathbf{x},\mathbf{y})$ means that there is no loop between the two. Using this, the ordinary predicates in a rule r can be ordered as follows:

$$\delta(m_i, r, S) = \bigwedge_{a(\mathbf{x})\in H(r)} \; \bigwedge_{b\in Int(m_i)} \; \bigwedge_{b(\mathbf{z})\in B^+_o(r)} D^{SS}_{ba}(\mathbf{z},\mathbf{x}) \wedge \neg D^{SS}_{ab}(\mathbf{x},\mathbf{z}) \ .$$

Concerning module atoms, we upgrade the translation for module atoms μ' to $\mu^\star$ with atom $a(\mathbf{x})$ as an additional argument. This new translation not only takes care of matching labels but also prevents loops between the output atom of β and $a(\mathbf{x})$, as well as loops between input predicates and formal arguments of the respective module call. In the following formula, p_k and $q_{j,k}$ come correspondingly from the input predicate list $\mathbf{p}$ of β and the formal arguments $\mathbf{q}_j$ of module m_j:

$$\begin{aligned}\mu^\star(m_i, \beta, S, a(\mathbf{x})) = \bigvee_{T\subseteq HB_{\mathbf{P}}|_{\mathbf{q}_j}} & \beta_{m_i,S,T} \wedge o^T(\mathbf{y}) \wedge D^{TS}_{oa}(\mathbf{y},\mathbf{x}) \wedge \neg D^{ST}_{ao}(\mathbf{x},\mathbf{y}) \wedge \\ & \forall \mathbf{z} \left(\bigwedge D^{ST}_{p_k q_{j,k}}(\mathbf{z},\mathbf{z}) \wedge \neg D^{TS}_{q_{j,k} p_k}(\mathbf{z},\mathbf{z}) \right) \ .\end{aligned}$$

The right hand side of the completion applies to every intensional predicate. Intuitively, this formula makes sure that whenever a head is true, then there must be some rule with the body satisfied, plus there is no loop involving the head and any atom in the body (both ordinary and module atoms), or between the input predicates and the corresponding formal input parameters of the called module; this is encoded in δ and $\mu^\star$, respectively:

$$\begin{aligned}&\rho(m_i, a(\mathbf{x}), S) = \\ &\forall \mathbf{x} \left[a^S(\mathbf{x}) \supset \bigvee_{r\in SR(a(\mathbf{x}),R_i)} \exists \mathbf{y}_1, \ldots, \mathbf{y}_n \left(\bigwedge_{p_l(\mathbf{y}_l)\in B^+_o(r)} p^S_l(\mathbf{y}_l) \wedge \bigwedge_{p_l(\mathbf{y}_l)\in B^-_o(r)} \neg p^S_l(\mathbf{y}_l) \wedge \right.\right. \\ &\qquad \left.\left. \delta(m_i, r, S) \wedge \bigwedge_{\beta\in B^+_m(r)} \mu^\star(m_i, \beta, S, a(\mathbf{x})) \wedge \bigwedge_{\beta\in B^-_m(r)} \overline{\mu'}(m_i, \beta, S) \right) \right] .\end{aligned}$$

Example 14. Continuing with Example 13, we get

$$\begin{aligned}\rho(m_1, p, \emptyset) = p^{S_1} \supset & (\neg p^{S_1} \wedge r^{S^0_2} \wedge D^{S^0_2 S_1}_{rp} \wedge \neg D^{S_1 S^0_2}_{pr} \wedge D^{S_1 S^0_2}_{pq} \wedge \neg D^{S^0_2 S_1}_{qp}) \\ & \vee (p^{S_1} \wedge r^{S^1_2} \wedge D^{S^1_2 S_1}_{rp} \wedge \neg D^{S_1 S^1_2}_{pr} \wedge D^{S_1 S^1_2}_{pq} \wedge \neg D^{S^1_2 S_1}_{qp}).\end{aligned}$$

Then, the ordered completion for an intensional predicate a is simply the conjunction of γ' and ρ:

$$\psi(m_i, a(\mathbf{x}), S) = \gamma'(m_i, a(\mathbf{x}), S) \wedge \rho(m_i, a(\mathbf{x}), S).$$

The ordered completion for a value call $P_i[S]$ is the collection of ordered completions of all intensional predicates and the realization of the input via predicates in $\mathbf{q}_i$, all labeled with S:

$$\psi(\mathbf{P}, P_i[S]) = \bigwedge_{a \in Int(m_i)} \psi(m_i, a(\mathbf{x}), S) \wedge \bigwedge_{\chi^S(q_{i,j}(\mathbf{c}))=1} q^S_{i,j}(\mathbf{c}) \ .$$

The only thing left is to capture the closure condition of the dependencies D^{ST}_{qp}, not only inside but also across module instances. In the formula below, we consider triples of value calls $P_i[S]$, $P_j[T]$, and $P_k[U]$ (not necessarily distinct) coming from the call graph $VC(\mathbf{P})$. Then,

$$\begin{aligned}&\tau(P_i[S], P_j[T], P_k[U]) = \\ &\bigwedge_{p \in Int(m_i)} \bigwedge_{q \in Int(m_j)} \bigwedge_{r \in Int(m_k)} \forall \mathbf{xyz}(D^{ST}_{pq}(\mathbf{x},\mathbf{y}) \wedge D^{TU}_{qr}(\mathbf{y},\mathbf{z}) \supset D^{SU}_{pr}(\mathbf{x},\mathbf{z})).\end{aligned}$$

Finally, the ordered completion of the whole MLP $\mathbf{P}$ is given by collecting the completions for all value calls in the call graph $VC(\mathbf{P})$ and the closure axiom of the dependency ordering between labeled predicates,

$$\tau(\mathbf{P}) = \bigwedge_{P_i[S], P_j[T], P_k[U] \in VC(\mathbf{P})} \tau(P_i[S], P_j[T], P_k[U]) \ ,$$

i.e.,

$$\mathit{\Omega}(\mathbf{P}) = \tau(\mathbf{P}) \wedge \bigwedge_{P_i[S] \in VC(\mathbf{P})} \psi(\mathbf{P}, P_i[S]) \ ,$$

Example 15. The formulas in Examples 13 and 14 give us the encoding $\psi(\mathbf{P}, P_1[]) = \gamma'(m_1, p, \emptyset) \wedge \rho(m_1, p, \emptyset)$ for module m_1 of the MLP $\mathbf{P}$ in Example 1. For m_2, we have:

- $\psi(\mathbf{P}, P_2[\emptyset]) = (q^{S^0_2} \supset r^{S^0_2}) \wedge (r^{S^0_2} \supset q^{S^0_2} \wedge D^{S^0_2 S^0_2}_{qr} \wedge \neg D^{S^0_2 S^0_2}_{rq})$,
- $\psi(\mathbf{P}, P_2[\{q\}]) = q^{S^1_2} \wedge (q^{S^1_2} \supset r^{S^1_2}) \wedge (r^{S^1_2} \supset q^{S^1_2} \wedge D^{S^1_2 S^1_2}_{qr} \wedge \neg D^{S^1_2 S^1_2}_{rq})$,
- $\tau(\mathbf{P}) = \bigwedge D^{T_1 T_2}_{p_1 p_2} \wedge D^{T_2 T_3}_{p_2 p_3} \supset D^{T_1 T_3}_{p_1 p_3}$, where $p_i \in \{p, q, r\}$; $T_i = S_1$ if $p_i = p$ and $T_i \in \{S^0_2, S^1_2\}$ otherwise.

The ordered completion $\mathit{\Omega}(\mathbf{P}) = \tau(\mathbf{P}) \wedge \psi(\mathbf{P}, P_1[]) \wedge \psi(\mathbf{P}, P_2[\emptyset]) \wedge \psi(\mathbf{P}, P_2[\{q\}])$ has a single model whose projection to labeled atoms is $\{r^{S^1_2}, q^{S^1_2}\}$. This model corresponds to the answer set mentioned in Example 1.

The following theorem shows the correctness of our translation. For this, given an MLP $\mathbf{P}$ and an H-structure $\mathfrak{M}$, we define the *derivation ordering* of $\mathbf{P}$ wrt. $\mathfrak{M}$ as the set $D^{\mathfrak{M}}(\mathbf{P})$ of all facts $D^{TS}_{qp}(\mathbf{c}_2, \mathbf{c}_1)$ such that (i) there exists a path from $p(\mathbf{c}_1)$ to $q(\mathbf{c}_2)$ in the modular dependency graph $MG_{\mathbf{P}}$, (ii) $\mathbf{c}_1 \in p^{\mathfrak{M},S}$ as p is an intensional predicate, and (iii) $\mathbf{c}_2 \in q^{\mathfrak{M},T}$ if q is an intensional predicate or $\mathbf{c}_2 \in q^{\mathfrak{M}}$ if q is an extensional predicate, where $T \in VC(\mathbf{P})$.

Theorem 2. *Let $\mathbf{P} = (m_1, \ldots, m_n)$ be an MLP in which all negated module atoms are monotonic. Then, (i) if an H-structure $\mathfrak{M}$ for $\mathbf{P}$ is an answer set of $gr(\mathbf{P}, \mathfrak{M})$, then $M \cup D(\mathbf{P})$ is a model of $\Omega(\mathbf{P})$, where*

$$M = \bigcup_{P_i[S] \in VC(\mathbf{P})} \{q^S(\mathbf{c}) \mid q \in \mathit{Ext}(m_i) \wedge \mathbf{c} \in q^{\mathfrak{M}}\} \cup \{p^S(\mathbf{c}) \mid p \in \mathit{Int}(m_i) \wedge \mathbf{c} \in p^{\mathfrak{M},S}\};$$

(ii) if M is a finite model of $\Omega(\mathbf{P})$, then the H-structure $\mathfrak{M}$ for $\mathbf{P}$ where (a) for each extensional predicate q, $q^{\mathfrak{M}} = \{\mathbf{c} \mid P_i[S] \in VC(\mathbf{P}) \wedge q^S(\mathbf{c}) \in M\}$, and (b) for each intensional predicate p in module m_i with input S, $p^{\mathfrak{M},S} = \{\mathbf{c} \mid P_i[S] \in VC(\mathbf{P}) \wedge p^S(\mathbf{c}) \in M\}$, is an answer set of $gr(\mathbf{P}, \mathfrak{M})$.

6 Discussion

The translations $\Lambda(\mathbf{P})$ and $\Omega(\mathbf{P})$ from above allow us to express the existence of answer sets of an MLP $\mathbf{P}$ as a satisfiability problem in propositional respectively predicate logic that is decidable; however, for arbitrary call-by-value, the resulting formulas are huge in general, given that there are double exponential many value calls $P_i[S]$ for an input $\mathbf{q}$ in general. Furthermore, loops can be very long; in the general case, they can have double exponential length. However, the intrinsic complexity of MLPs already mentioned in Section 4 suggests that even in the propositional case (where the number of different inputs S is at most exponential) we can not expect a polynomially computable transformation of brave inference $\mathbf{P} \models a$ into a propositional SAT instance, as the problem is EXP-complete for propositional Horn MLPs and NEXP-complete for propositional normal MLPs.

The ordered completion formula $\Omega(\mathbf{P})$, which can be seen as a Σ_1^1 formula over a finite structure and is thus evaluable in nondeterministic exponential time. Here, in the propositional case the input values S and T may be encoded using (polynomially many) predicate arguments (e.g., $o^T(\mathbf{y})$ becomes $o(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} = x_1, \ldots, x_k$ encodes T) and disjunction/conjunction over S and T expressed by (first-order) quantification. In this way, it is possible to obtain a Σ_1^1 formula of polynomial size over a finite structure, such that this modified transformation is worst-case optimal with respect to the complexity of propositional normal MLPs. Similar encoding techniques can be applied for non-ground MLPs if the predicate arities of formal input predicates are bounded by a constant.

In the general non-ground case, such polynomial encoding techniques are not evident; already in the Horn case deciding $\mathbf{P} \models a$ is 2EXP-complete, and for normal MLPs brave inference is 2NEXP-complete. One may resort to predicate variables for encoding S and T, and naturally arrive at a formula in higher-order logic (e.g., $o^T(\mathbf{y})$ becomes $o(\mathbf{T}, \mathbf{y})$ where $\mathbf{T} = T_1, \ldots, T_k$ is a list of predicate variables for the formal input predicates $\mathbf{q} = q_1, \ldots, q_k$). It remains to be seen, however, whether the structure of the resulting (polynomial-size) formula would readily permit worst-case optimal evaluation with respect to the complexity of MLPs.

Noticeably, however, we do not get a blowup if no call-by-value is made, i.e., if all inputs lists are empty (which means all S and T have the single value $\emptyset$). This setting is still useful for structured programming, and amounts in the propositional case to the

DLP-functions of [15] (but in contrast permits unlimited recursion through modules, in particular positive recursion). Our results thus also provide ordered completion formulas for DLP-functions over normal programs.

7 Conclusion

In this paper, we have studied encodings of answer sets of Modular Nonmonotonic Logic Programs (MLPs) into first-order formulas, in the line of recent work in Answer Set Programming.

As for future work, refinement of the results and exploitation of the results for answer sets computation using SAT and QBF solvers, as well as theorem provers remains to be investigated; here, fragments of MLPs that allow for reasonable encodings might be considered, and the suitability of higher-order theorem provers evaluated.

For this paper, we have considered as context C the set $VC(\mathbf{P})$ of all value calls, which thus can be omitted. Intuitively, a given context C may be incorporated by ensuring that loop formulas are built only for relevant instantiation signatures $(\mathcal{S}_1, \ldots, \mathcal{S}_n)$; for modular loops, which are those that contain some value call $\mathcal{S}_i$ inside C; furthermore, either none or all value calls $\mathcal{S}_i$ must be in C. Relative to an interpretation $\mathbf{M}$, the minimal context $C = V(CG_{\mathbf{P}}(\mathbf{M}))$ may be defined using suitable predicates respective propositions. The technical elaboration of these ideas is beyond this paper.

Another assumption that we have made was that module atoms under negation are monotonic, in order to readily apply loop formula techniques despite the semantics of MLPs based on the FLP-reduct. It would be interesting to look into the full case of MLPs with arbitrary module atoms. Recently, unfounded sets for logic programs with arbitrary aggregates have been defined in [10]. Given that for ordinary logic programs unfounded sets are a semantic counterpart of loop formulas, this may inspire a similar notion of unfounded set for MLPs and help developing a syntactic counterpart in terms of loop formulas. The papers [17, 21], which inspect the FLP-semantics on a more principled level, may also be useful in this respect. Different from answer semantics under the GL-reduct, not only positive atoms need to be considered for derivability, but also negated non-montonic module atoms.

A further issue are encodings for disjunctive MLPs, i.e., MLPs where the head of a rule may be a disjunction $\alpha_1 \vee \cdots \vee \alpha_k$ of atoms. Loop formulas for ordinary disjunctive logic programs have been developed in [16], and for general propositional theories under Answer Set Semantics in [12]. There is no principal obstacle to extend the loop formula encoding of this paper to disjunctive MLPs, and doing this is routine. In contrast, ordered completion formulas for disjunctive MLPs and already ordinary LPs needs further work; they may require a blowup given that ordinary disjunctive Datalog programs have $\mathrm{NEXP}^{\mathrm{NP}}$ complexity.

Finally, relationships to other semantics of logic programming is an interesting issue. Chen et al. [4] showed that loops with at most one external support rule in the program have a close connection to (disjunctive) well-founded semantics. Studying MLPs under similar restrictions could provide similar results, yet well-founded semantics for MLPs remains to be formalized.

References

1. Apt, K., Blair, H., Walker, A.: Towards a Theory of Declarative Knowledge. In: Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan Kaufman (1988)
2. Arni, F., Ong, K., Tsur, S., Wang, H., Zaniolo, C.: The deductive database system $\mathcal{LDL}$++. Theor. Pract. Log. Prog. 3(1), 61–94 (2003)
3. Asuncion, V., Lin, F., Zhang, Y., Zhou, Y.: Ordered completion for first-order logic programs on finite structures. In: AAAI'10. pp. 249–254. AAAI Press (2010)
4. Chen, X., Ji, J., Lin, F.: Computing loops with at most one external support rule for disjunctive logic programs. In: ICLP'09. pp. 130–144. Springer (2009)
5. Clark, K.L.: Negation as failure. In: Logic and Data Bases. pp. 293–322 (1978)
6. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular Nonmonotonic Logic Programming Revisited. In: ICLP'09. pp. 145–159. Springer (2009)
7. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM T. Database Syst. 22(3), 364–417 (1997)
8. Eiter, T., Gottlob, G., Veith, H.: Modular Logic Programming and Generalized Quantifiers. In: LPNMR'97. pp. 290–309. Springer (1997), extended paper CD-TR 97/108, Institut f. Informationssysteme, TU Wien 1997
9. Eiter, T., Leone, N., Saccà, D.: On the Partial Semantics for Disjunctive Deductive Databases. Ann. Math. Artif. Intell. 19(1/2), 59–96 (1997)
10. Faber, W.: Unfounded sets for disjunctive logic programs with arbitrary aggregates. In: LPNMR'05. pp. 40–52. Springer (2005)
11. Faber, W., Leone, N., Pfeifer, G.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. 175(1), 278–298 (2011)
12. Ferraris, P., Lee, J., Lifschitz, V.: A generalization of the Lin-Zhao theorem. Ann. Math. Artif. Intell. 47(1-2), 79–101 (2006)
13. Gaifman, H., Shapiro, E.: Fully abstract compositional semantics for logic programs. In: POPL'89. pp. 134–142. ACM (1989)
14. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generat. Comput. 9(3–4), 365–385 (1991)
15. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity Aspects of Disjunctive Stable Models. J. Artif. Intell. Res. 35, 813–857 (2009)
16. Lee, J., Lifschitz, V.: Loop formulas for disjunctive logic programs. In: ICLP'03. pp. 451–465. Springer (2003)
17. Lee, J., Meng, Y.: On reductive semantics of aggregates in answer set programming. In: LPNMR'09. pp. 182–195. Springer (2009)
18. Lifschitz, V., Turner, H.: Splitting a Logic Program. In: ICLP'94. pp. 23–38. MIT-Press (1994)
19. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artif. Intell. 157(1-2), 115–137 (2004)
20. Ross, K.: Modular Stratification and Magic Sets for Datalog Programs with Negation. J. ACM 41(6), 1216–1267 (1994)
21. Truszczyński, M.: Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. Artif. Intell. 174(16–17), 1285–1306 (2010)