Decidability and implementation of parametrized logic programs

Ricardo Gonçalves and José Júlio Alferes*

CENTRIA - Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Abstract. Parametrized logic programs are very expressive logic programs that generalize normal logic programs under the stable model semantics, by allowing complex formulas of a parameter logic to appear in the body and head of rules. In this paper we study the decidability of these rich programs and propose an implementation that combines, in a modular way, a reasoner for the parameter logic with an answer set solver.

1 Introduction

Parametrized logic programming [9] was introduced as an extension of answer set programming [8] with the motivation of providing a meaning to theories combining both logic programming connectives with other logical connectives, and allowing complex formulas using these connectives to appear in the head and body of a rule. The main idea is to fix a monotonic logic $\mathcal{L}$, called the parameter logic, and build up logic programs using formulas of $\mathcal{L}$ instead of just atoms. The obtained parametrized logic programs have, therefore, the same structure of normal logic programs, the only difference being the fact that atomic symbols are replaced by formulas of $\mathcal{L}$.

When applying this framework, the choice of the parameter logic depends on the domain of the problem to be modeled. As examples, [9] shows how to obtain the answer-set semantics of logic programs with explicit negation, a paraconsistent version of it, and also the semantics of MKNF hybrid knowledge bases [15], using an appropriate choice of the parameter logic. Moreover, [10] introduces deontic logic programs using standard deontic logic [20] as the parameter logic.

Parametrized logic programming can be seen as a framework which allow us to add non-monotonic rule based reasoning on top of an existing (monotonic) language. This view is quite interesting, in particular in those cases where we already have a monotonic logic to model a problem, but we are still lacking some conditional or non-monotonic reasoning. In these situations, parametrized logic programming offers a modular framework for adding such conditional and non-monotonic reasoning, without having to give up on the monotonic logic at

* The first author was supported by FCT under the postdoctoral grant SFRH/BPD/47245/2008. The work was partially supported by projects ERRO – PTDC/EIA-CCO/121823/2010, and ASPEN – PTDC/EIA-CCO/110921/2009.

hand. One interesting example is the case of MKNF hybrid knowledge bases, where the existing monotonic logics are description logics.

However, in order to make parametrized logic programming usable in practice, we need to prove that this rich combination does not compromise decidability in the case of a decidable parameter logic. Moreover, given a decidable parameter logic, for pragmatic reasons the implementation for a reasoner should make a modular use of an existing reasoner for the parameter logic and an answer set solver. This modularity is extremely important since it allows us to use the large body of successful research done in the area of stable model semantics implementation and answer set programming.

In this paper, after introducing the framework of parametrized logic programs (Section 2), we address the decidability of the stable model entailment of parametrized logic programs and study the implementation of a reasoner for parametrized logic programs, combining a reasoner for the parameter logic and answer set solver in a modular way (Section 3). We also study some interesting examples of parameter logics over a restricted language that have better computational properties than the general case. We end with some conclusions and draw some paths for future research (Section 4).

2 Parametrized logic programs

In this section we introduce the syntax and semantics of normal parametrized logic programs [9].

2.1 Language

The syntax of a normal parametrized logic program has the same structure of that of a normal logic program. The only difference is that the atomic symbols of a normal parametrized logic program are replaced by formulas of a parameter logic, which is restricted to be a monotonic logic. Let us start by introducing the necessary concepts related with the notion of (monotonic) logic.

Definition 1. *A (monotonic) logic is a pair $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ where L is a set of formulas and $\vdash_{\mathcal{L}}$ is a Tarskian consequence relation [21] over L, i.e., satisfying the following conditions, for every $T \cup \Phi \cup \{\varphi\} \subseteq L$,*
Reflexivity: *if $\varphi \in T$ then $T \vdash_{\mathcal{L}} \varphi$;*
Cut: *if $T \vdash_{\mathcal{L}} \varphi$ for all $\varphi \in \Phi$, and $\Phi \vdash_{\mathcal{L}} \psi$ then $T \vdash_{\mathcal{L}} \psi$;*
Weakening: *if $T \vdash_{\mathcal{L}} \varphi$ and $T \subseteq \Phi$ then $\Phi \vdash_{\mathcal{L}} \varphi$.*

When clear from the context we write $\vdash$ instead of $\vdash_{\mathcal{L}}$. Let $Th(\mathcal{L})$ be the set of *logical theories of* $\mathcal{L}$, i.e. the set of subsets of L closed under the relation $\vdash_{\mathcal{L}}$. One fundamental characteristic of the above definition is that, for every (monotonic) logic $\mathcal{L}$, the tuple $\langle Th(\mathcal{L}), \subseteq \rangle$ is a complete lattice with smallest element the set $Theo = \{\varphi \in L : \emptyset \vdash \varphi\}$ of theorems of $\mathcal{L}$ and greatest element the set L of all formulas of $\mathcal{L}$. Given a subset A of L we denote by $A^{\vdash_{\mathcal{L}}}$ the smallest logical theory of $\mathcal{L}$ that contains A, and call it *the logical theory generated by A in $\mathcal{L}$.*

In the following we consider fixed a (monotonic) logic $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ and call it the *parameter logic*. The formulas of $\mathcal{L}$ are dubbed *(parametrized) atoms* and a *(parametrized) literal* is either a parametrized atom φ or its negation *not* φ, where *not* denotes default negation. *Default literals* are those of the form *not* φ.

Definition 2. *A* normal $\mathcal{L}$ parametrized logic program *is a set of rules*

$$\varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \delta_1, \ldots, not\ \delta_m \tag{1}$$

where $\varphi, \psi_1, \ldots, \psi_n, \delta_1, \ldots, \delta_m \in L$.

A definite $\mathcal{L}$ parametrized logic program *is a set of rules without negations as failure, i.e. of the form* $\varphi \leftarrow \psi_1, \ldots, \psi_n$ *where* $\varphi, \psi_1, \ldots, \psi_n \in L$.

As usual, the symbol $\leftarrow$ represents rule implication, the symbol "," represents conjunction and the symbol *not* represents default negation. A rule as (1) has the usual reading that φ should hold whenever $\psi_1, \ldots, \psi_n$ hold and $\delta_1, \ldots, \delta_m$ are not known to hold. If $n = 0$ and $m = 0$ then we just write $\varphi \leftarrow$.

Given a rule r of the form (1), we define $head(r) = \varphi$, $body^+(r) = \{\psi_1, \ldots, \psi_n\}$, $body^-(r) = \{\delta_1, \ldots, \delta_m\}$ and $body(r) = body^+(r) \cup body^-(r)$. Given a parametrized logic program $\mathcal{P}$ we define $form(\mathcal{P})$ to be the set of all formulas of the parameter language L appearing in $\mathcal{P}$, i.e., $form(\mathcal{P}) = \bigcup_{r \in \mathcal{P}} (\{head(r)\} \cup body(r))$. We also define the set $head(\mathcal{P}) = \{head(r) : r \in \mathcal{P}\}$.

2.2 Semantics

The semantics of parametrized logic programs is defined as a generalization of the stable model semantics [8] of normal logic programs.

In the normal logic programs, an interpretation is just a set of atoms. In a parametrized logic program, since we substitute atoms by formulas of a parameter logic, the first idea is to take sets of formulas of the parameter logic as interpretations. The problem is that, contrary to the case of atoms, the parametrized atoms are not independent of each other. This interdependence is governed by the consequence relation of the parameter logic. For example, if we take classical propositional logic (CPL) as the parameter logic, we have that if the parametrized atom $p \wedge q$ is true then so are the parametrized atoms p and q. If we take, for example, standard deontic logic SDL [20] as parameter, we have that, since $\mathbf{O}(p \vee q), \mathbf{O}(\neg p) \vdash_{SDL} \mathbf{O}(q)$, any SDL logical theory containing both $\mathbf{O}(p \vee q)$ and $\mathbf{O}(\neg p)$ also contains $\mathbf{O}(q)$.

To account for this interdependence, we use logical theories (sets of formulas closed under the consequence of the logic) as the generalization of interpretations, thus capturing the above mentioned interdependence.

Definition 3. *A (parametrized) interpretation is a logical theory of* $\mathcal{L}$.

Definition 4. *An interpretation* T *satisfies a rule*

$$\varphi \leftarrow \psi_1, \ldots, \psi_n, not\ \delta_1, \ldots, not\ \delta_m$$

if $\varphi \in T$ *whenever* $\psi_i \in T$ *for every* $i \in \{1, \ldots, n\}$ *and* $\delta_j \notin T$ *for every* $j \in \{1, \ldots, m\}$.

An interpretation is a model of logic program $\mathcal{P}$ if it satisfies every rule of $\mathcal{P}$. We denote by $Mod_{\mathcal{L}}(P)$ the set of models of $\mathcal{P}$.

The ordering over interpretations is the usual one: If T_1 and T_2 are two interpretations then we say that $T_1 \leq T_2$ if $T_1 \subseteq T_2$. Moreover, given such ordering, minimal and least interpretations may be defined in the usual way.

As in the case of non parametrized programs, we start by assigning semantics to definite parametrized programs. Recall that the stable model of a definite logic program is its least model. In order to generalize this definition to the parametrized case we need to establish that the least parametrized model exists for every definite $\mathcal{L}$ parametrized logic program.

Theorem 1 ([9]). *Every definite $\mathcal{L}$ parametrized logic program has a least model, denoted by $S_P^{\mathcal{L}}$.*

Note that this theorem holds for every choice of the parameter logic $\mathcal{L}$.

The stable model semantics of a normal $\mathcal{L}$ parametrized logic programs is defined using a Gelfond-Lifschitz like operator.

Definition 5. *Let $\mathcal{P}$ be a normal $\mathcal{L}$ parametrized logic program and T an interpretation. The GL-transformation of $\mathcal{P}$ modulo T is the program $\frac{P}{T}$ obtained from P by performing the following operations:*

- *remove from $\mathcal{P}$ all rules which contain a literal not φ such that $T \vdash_{\mathcal{L}} \varphi$;*
- *remove from the remaining rules all default literals.*

Since $\frac{P}{T}$ is a definite $\mathcal{L}$ parametrized program, it has an unique least model J. We define $\Gamma(T) = J$.

Stable models of a parametrized logic program are then defined as fixed points of this Γ operator.

Definition 6. *An interpretation T of an $\mathcal{L}$ parametrized logic program $\mathcal{P}$ is a stable model of $\mathcal{P}$ iff $\Gamma(T) = T$. A formula φ is true under the stable model semantics, denoted by $\mathcal{P} \models_{SM} \varphi$ iff it belongs to all stable models of $\mathcal{P}$. We denote by $SM(\mathcal{P})$ the set of all stable models of $\mathcal{L}$.*

2.3 Examples

Parametrized logic programs are very general and flexible, allowing not only to capture well-known extensions of the stable model semantics of normal logic programs, but also to extend them further. In [9] it is shown that normal logic programs and extended logic programs correspond to an appropriate choice of the parameter logic.

One interesting case that already goes beyond the usual extensions of normal logic programs is to use a parameter logic over a full propositional language. Note that this is different, and in fact orthogonal, to the so-called nested logic programs [6]. Nested logic programs are propositional combinations of the logic programming connectives. In the case of parametrized logic programs, propositional nesting only appears at the level of the atoms.

Example 1 (Propositional logic programs). Let us now consider a full propositional language L built over a set $\mathcal{P}$ of propositional symbols using the usual connectives $(\neg, \vee, \wedge, \Rightarrow)$. Many consequence relations can be defined over this language. We present three interesting examples: classical logic, Belnap's paraconsistent logic [2] and intuitionistic logic. Consider the following programs:

$$P_1 \begin{cases} p \leftarrow \neg q \\ p \leftarrow q \end{cases} \qquad P_2 \begin{cases} p \leftarrow \neg q \vee q \end{cases} \qquad P_3 \begin{cases} q \leftarrow \\ (q \vee s) \Rightarrow p \leftarrow \\ r \leftarrow p \end{cases}$$

$$P_4 \begin{cases} r \leftarrow \\ \neg p \leftarrow \\ (p \vee q) \leftarrow r \\ s \leftarrow q \end{cases} \qquad P_5 \begin{cases} p \leftarrow not\ q, not\ \neg q \end{cases} \qquad P_6 \begin{cases} p \leftarrow not\ (q \vee \neg q) \end{cases} \qquad P_7 \begin{cases} p \leftarrow \\ \neg p \leftarrow \end{cases}$$

Let $\mathcal{L} = \langle L, \vdash_{CPL} \rangle$ be Classical Propositional Logic (CPL) over the language L. Let us study in detail the semantics of P_1. Note that every CPL logical theory that does not contain neither p nor $\neg p$ satisfies P_1. In particular, the set $Taut$ of tautologies of CPL is a model of P_1. So, $S^{CPL}_{P_1} = Taut$. This means that $p, \neg p, q, \neg q \notin S^{CPL}_{P_1}$. We also have that $S^{CPL}_{P_2} = \{p\}^{\vdash ails}$. So, in the case of P_2 we have that $p \in S^{CPL}_{P_2}$. Also, we have that $r \in S^{CPL}_{P_3}$ and $s \in S^{CPL}_{P_4}$.

In the case of P_5 its stable models are the CPL logical theories that contain p and do not contain q nor $\neg q$. Therefore, we have that $p \in S^{CPL}_{P_5}$. In the case of P_6, since $(p \vee \neg p) \in T$ for every CPL logical theory T we can conclude that the only stable model of P_6 is the set $Theo$ of theorems of CPL. Therefore $p \notin S^{CPL}_{P_6}$. Regarding P_7, it is clear that $S^{CPL}_{P_7}$ equals the (inconsistent) set of all formulas. Note that, like in answer-sets, stable models of parametrized programs can be inconsistent, this being conceptually different from the case when there are no answer-sets.

Consider now $\mathcal{L} = \langle L, \vdash_4 \rangle$ the 4-valued Belnap paraconsistent logic $Four$. Consider the program P_4. Contrarily to the case of CPL, in $Four$ it is not the case that $\neg p, (p \vee q) \vdash_4 q$. Therefore we have that $q, s \notin S^{Four}_{P_4}$.

Let now $\mathcal{L} = \langle L, \vdash_{IPL} \rangle$ be the Intuitionistic Propositional Logic IPL. It is well-known that $q \vee \neg q$ is not a theorem of IPL. Therefore, considering program P_2 we have $S^{IPL}_{P_2} = \emptyset^{\vdash_{IPL}}$. So, contrarily to the case of CPL, we have that $p \notin S^{IPL}_{P_2}$. Using the same idea for program P_6 we can conclude, contrarily to the case of CPL, that $p \in S^{IPL}_{P_6}$.

Another interesting class of logic that can be taken as parameter are modal logics [5]. Modal logics are fundamental in many areas of Artificial Intelligence. They are quite flexible, expressive, and quite often decidable. By using parametrized logic programs with a modal logic as the parameter logic we are thus adding a non-monotonic layer to an already expressive language.

Example 2 (Modal logic).

Consider modal logic language L_m built over a set $\mathcal{P}$ of propositional symbols using the usual connectives $\neg, \vee, \wedge, \Rightarrow$ and the modal operators $\Box, \Diamond$. Let $\mathcal{L}_m = \langle L_m, \vdash_m \rangle$ be a modal logic over the language L_m, where the consequence relation

is obtained from usual Kripke style semantics. Of course, the particular modal logic we obtain depends on the restriction we impose in the Kripke models. Just to mention a few interesting examples, $\mathcal{L}_m$ could be epistemic logic, usually an S_5 modal logic, deontic logic, usually a KD modal logic and doxastic logic, usually a $KD45$ modal logic. Our aim with this example is just to stress that we can choose quite interesting and expressive logics as the parameter logic. Just to give an example, in [10, 11] a very rich non-monotonic framework for reasoning about normative systems can be obtained by choosing modal logic KD, also known as Standard Deontic Logic [20], as the parameter logic.

3 Decidability and implementation

We have seen how general is the construction of logic programs using a parameter logic. The question that naturally arises now is whether this combination of a monotonic logic and a non-monotonic framework preserves decidability. Moreover, even if decidability is preserved, there is still the question of whether we can use existing tools for the parameter logic together with an ASP solver to implement a reasoning tool for the combination. In this section we address both these issues. We first show that decidability is preserved if the parameter logic is decidable and then we also show how to combine an existing reasoner for a given parameter logic with an ASP solver.

We start with an interesting observation: even for logics over a propositional logical language built from a finite number of propositional symbols, the number of logical theories may be infinite. An immediate consequence is that the number of possible stable models of a finite parametrized logic program can be infinite. Interestingly, as we show below, decidability is not necessarily compromised. The key idea is that, given a finite parametrized logic program $\mathcal{P}$, we are able to prove that only those logical theories generated by sets of formulas appearing in $\mathcal{P}$ can be stable models of $\mathcal{P}$, and these are in a finite number.

Theorem 2. *Let $\mathcal{P}$ be a finite parametrized logic program. If T is a stable model of $\mathcal{P}$ then there exists $A \subseteq form(\mathcal{P})$ such that $T = A^{\vdash_{\mathcal{L}}}$.*

Proof. Let T be a stable model of $\mathcal{P}$. Consider the set $A = T \cap form(\mathcal{P})$, i.e., the restriction of T to the set of formulas appearing in $\mathcal{P}$. Since T is a logical theory of $\mathcal{L}$, $A \subseteq T$ and $\mathcal{L}$ is monotonic, we have that $A^{\vdash_{\mathcal{L}}} \subseteq T^{\vdash_{\mathcal{L}}} = T$. We aim to prove that, in fact, $A^{\vdash_{\mathcal{L}}} = T$. Since A is the restriction of T to the formulas of $\mathcal{P}$ we have that $\frac{\mathcal{P}}{A^{\vdash_{\mathcal{L}}}} = \frac{\mathcal{P}}{T}$. Then, we have that $A^{\vdash_{\mathcal{L}}}$ is also a model of $\frac{\mathcal{P}}{T}$. Since T is a stable model of $\mathcal{P}$ it is the minimal model of $\frac{\mathcal{P}}{T}$. Therefore, we can conclude $T \subseteq A^{\vdash_{\mathcal{L}}}$, which then implies that $T = A^{\vdash_{\mathcal{L}}}$. □

The above theorem has as immediate consequence the fact that every finite parametrized logic program has a finite number of stable models.

Corollary 1. *Let $\mathcal{P}$ be a finite parametrized logic program. Then, $\mathcal{P}$ has finitely many stable models.*

With this, we can now prove the decidability result.

Theorem 3. *Let $\mathcal{P}$ be a finite parametrized logic program over a decidable parameter logic $\mathcal{L}$ and φ a formula of $\mathcal{L}$ (not necessarily in $\mathcal{P}$). Then, it is decidable the problem of checking if $\mathcal{P} \models_{SM} \varphi$ is the case.*

Proof. First of all, note that we are assuming that $\mathcal{L}$ is a decidable logic, i.e., the problem of checking $\Phi \vdash_{\mathcal{L}} \varphi$, for a finite set Φ of $\mathcal{L}$ formulas, is decidable. Note also that the sets $form(\mathcal{P})$ and its subset $head(\mathcal{P})$ are finite.

Let us now introduce some necessary notation. Given a subset A of $form(\mathcal{P})$ we write $C(A)$ to denote its closure under $\mathcal{L}$ consequence, i.e., $C(A) = A^{\vdash_{\mathcal{L}}} \cap form(\mathcal{P})$. Given a subset A of $form(\mathcal{P})$, we can easily construct $C(A)$ by checking, for each $\psi \in form(\mathcal{P}) \setminus A$, if $A \vdash_{\mathcal{L}} \psi$;

In Fig. 1 we sketch an algorithm showing the decidability of the problem $\mathcal{P} \models_{\mathrm{SM}} \varphi$. It is based on the Gelfond-Lifschitz transformation with the additional use of an $\mathcal{L}$ oracle. The fundamental tool supporting the algorithm is the result in Theorem 2, since it restricts severely the number of $\mathcal{L}$ theories we need to check. To cut even more the number of theories to be checked we also use the well-known result in the logic programming area: a stable model of a normal logic program is always a subset of the set of heads of rules of the program. □

---

input: finite PLP $\mathcal{P}$ and $\mathcal{L}$ formula φ
  for each $\Phi \subseteq head(\mathcal{P})$ compute $C(\Phi)$
   if $\Phi = C(\Phi)$ then
    compute $\frac{\mathcal{P}}{\Phi}$
    compute $least(\frac{\mathcal{P}}{\Phi})$ restricted to $form(\mathcal{P})$:
      define $A_0 := C(\{\varphi : \varphi \leftarrow \in \mathcal{P}\})$
      compute $A_{i+1} := C(\{\varphi : \varphi \leftarrow \psi_1, \ldots, \psi_n \in \mathcal{P}$ and $\{\psi_1, \ldots, \psi_n\} \subseteq A_i\})$
       until $A_k = A_{k+1}$ for some k
       then set $least(\frac{\mathcal{P}}{\Phi}) := A_k$;
    if $least(\frac{\mathcal{P}}{\Phi}) = \Phi$ /* in this case $\Phi^{\vdash_{\mathcal{L}}}$ is a stable model of $\mathcal{P}$ */
       check if $\Phi \vdash_{\mathcal{L}} \varphi$
  if $\Phi \vdash_{\mathcal{L}} \varphi$ for every $\Phi \subseteq head(\mathcal{P})$ such that $\Phi^{\vdash_{\mathcal{L}}}$ is a stable model of $\mathcal{P}$,
   then $\mathcal{P} \models_{\mathrm{SM}} \varphi$ is the case.

Fig. 1. Decidability algorithm

The algorithm in the proof of Theorem 3 is interesting since it is a (basic) stable model like algorithm which, when necessary, makes queries to an $\mathcal{L}$-oracle. This makes it modular with respect to the $\mathcal{L}$-reasoner and it minimizes the calls to the $\mathcal{L}$-oracle. The algorithm has, nevertheless, a major drawback: it is not modular from the point of view of calculating stable models, in the sense that we cannot use existing ASP solvers to compute the stable models of a parametrized logic program. This modularity is extremely important since it would allow us to use

the large body of successful research done in the area of stable model semantics implementation and answer set programming. Our aim is precisely to propose an implementation of a reasoner for parametrized logic programs which modularly combines an ASP solver (such as Clasp [7]) with a reasoner for the parameter logic (such as the KED SDL solver [1] in the case of Standard Deontic Logic [20], or the HermiT [16] reasoner in the case SROIQ description logic [13]).

We start by proving a theorem that sets the ground for the construction of the modular reasoner. Consider a given parametrized logic program $\mathcal{P}$, and construct the following normal logic program $\mathcal{P}^N$ from $\mathcal{P}$:

$$\begin{aligned}\mathcal{P}^N = \mathcal{P} \cup \{\varphi \leftarrow \psi_1, \ldots, \psi_n : & \{\psi_1, \ldots, \psi_n\} \subseteq form(\mathcal{P}), \\ & \varphi \in form(\mathcal{P}) \backslash_{\{\psi_1, \ldots, \psi_n\}}, \\ & \{\psi_1, \ldots, \psi_n\} \vdash_{\mathcal{L}} \varphi\}.\end{aligned}$$

We call $\mathcal{P}^N$ the normal logic program obtained from $\mathcal{P}$, since the $\mathcal{L}$ formulas appearing in it are to be considered as normal logic programs atoms. The key idea underlying the construction of $\mathcal{P}^N$, in order to enforce the interdependency between the $\mathcal{L}$ formulas (which in $\mathcal{P}^N$ are just atoms), is to enrich $\mathcal{P}$ with rules that represent the possible reasoning in $\mathcal{L}$ occurring with the formulas of $\mathcal{P}$.

Since we are now considering usual normal logic programs, and to distinguish between the set of stable model of a parametrized logic program $\mathcal{P}$ (which is a subset of $2^{Th_{\mathcal{L}}}$) and the set of stable models of $\mathcal{P}$ viewed as a normal logic program (which is a subset of $2^{form(\mathcal{P})}$), we denote the latter by $AS(\mathcal{P})$. Note that $SM(\mathcal{P})$ and $AS(\mathcal{P})$ can be very different since AS does not take into account the interdependency between the (parametrized) atoms. As a simple example let $\mathcal{L}$ be a normal modal logic. Consider $\mathcal{P} = \{\Box p \leftarrow;\ \Box q \leftarrow \Box(p \vee r)\}$. Then, $SM(\mathcal{P}) = \{\{\Box p, \Box(p \vee r), \Box q\}^{\vdash_{\mathcal{L}}}\}$ but $AS(\mathcal{P}) = \{\{\Box p\}\}$.

Theorem 4. *Given a parametrized logic program $\mathcal{P}$, we have that*

$$SM(\mathcal{P}) = \{A^{\vdash_{\mathcal{L}}} : A \in AS(\mathcal{P}^N)\}$$

Proof. Let us start with some notation and a general comment. We use $\mathcal{P}^{\mathcal{L}}$ to denote the set of rules that are added to $\mathcal{P}$ in the definition of the program $\mathcal{P}^N$, i.e., $\mathcal{P}^N = \mathcal{P} \cup \mathcal{P}^{\mathcal{L}}$. Since the rules in $\mathcal{P}^{\mathcal{L}}$ represent sound consequences in $\mathcal{L}$, and since every logical theory of $\mathcal{L}$ is closed under $\mathcal{L}$ consequence, it follows immediately that every $\mathcal{L}$ logical theory satisfies all the rules in $\mathcal{P}^{\mathcal{L}}$.

We now prove the equality $SM(\mathcal{P}) = \{A^{\vdash_{\mathcal{L}}} : A \in AS(\mathcal{P}^N)\}$ by proving the two inclusions. Let us start by proving the left to right inclusion. Let T be a stable model of $\mathcal{P}$. We aim to prove that there exists a stable model A of $\mathcal{P}^N$ such that $T = A^{\vdash_{\mathcal{L}}}$. Take $A = T \cap form(\mathcal{P})$. We first prove that $A^{\vdash_{\mathcal{L}}} = T$. Since $\mathcal{L}$ is monotone, we have that $A^{\vdash_{\mathcal{L}}} \subseteq T^{\vdash_{\mathcal{L}}} = T$. To prove the reverse inclusion recall that T is the minimal $\mathcal{L}$ logical theory that satisfies $\frac{\mathcal{P}}{T}$. Since $A = T \cap form(\mathcal{P})$, it immediately follows that A satisfies $\frac{\mathcal{P}}{T}$. Therefore, $A^{\vdash_{\mathcal{L}}}$ is an $\mathcal{L}$ logical theory that satisfies $\frac{\mathcal{P}}{T}$. Since T is the minimal one, we have that $T \subseteq A^{\vdash_{\mathcal{L}}}$.

Now that we have proved that $A^{\vdash_{\mathcal{L}}} = T$, we need to prove that A is a stable model of $\mathcal{P}^N$. First of all, observe that $\frac{\mathcal{P}^N}{A} = \frac{\mathcal{P}}{A} \cup \mathcal{P}^{\mathcal{L}} = \frac{\mathcal{P}}{T} \cup \mathcal{P}^{\mathcal{L}}$. Let

$B \subseteq form(\mathcal{P})$ be a model of $\frac{\mathcal{P}^N}{A}$. Then, since B satisfies $\mathcal{P}^{\mathcal{L}}$, B is closed under $\mathcal{L}$ consequence. This in turn implies that $B^{\vdash_{\mathcal{L}}} \cap form(\mathcal{P}) = B$. Clearly $B^{\vdash_{\mathcal{L}}}$ is a model of $\frac{\mathcal{P}}{T}$, and, since T is the minimal $\mathcal{L}$ logical theory satisfying $\frac{\mathcal{P}}{T}$, we can conclude that $T \subseteq B^{\vdash_{\mathcal{L}}}$. But then $A = T \cap form(\mathcal{P}) \subseteq B^{\vdash_{\mathcal{L}}} \cap form(\mathcal{P}) = B$. Since this inclusion is the case for every B model of $\frac{\mathcal{P}^N}{A}$, we can conclude that A is the minimal model of $\frac{\mathcal{P}^N}{A}$, i.e., A is a stable model of $\mathcal{P}^N$.

We now prove the right to left inclusion. Let $A \subseteq form(\mathcal{P})$ be a stable model of $\mathcal{P}^N$. We aim to prove that $A^{\vdash_{\mathcal{L}}}$ is a stable model of $\mathcal{P}$. Since A is a stable model of $\mathcal{P}^N$ we have that A is the minimal model of $\frac{\mathcal{P}^N}{A} = \frac{\mathcal{P}}{A} \cup \mathcal{P}^{\mathcal{L}}$. Since A is a model of $\mathcal{P}^{\mathcal{L}}$ we have that A is closed under $\mathcal{L}$ consequence, i.e., $A^{\vdash_{\mathcal{L}}} \cap form(\mathcal{P}) = A$. We then have that $\frac{\mathcal{P}}{A^{\vdash_{\mathcal{L}}}} = \frac{\mathcal{P}}{A}$. Suppose there exists an $\mathcal{L}$ logical theory T such that $T \subset A^{\vdash_{\mathcal{L}}}$ and T satisfies $\frac{\mathcal{P}}{A^{\vdash_{\mathcal{L}}}}$. In that case, $T \cap form(\mathcal{P}) \subset A^{\vdash_{\mathcal{L}}} \cap form(\mathcal{P}) = A$ and $T \cap form(\mathcal{P})$ satisfies $\frac{\mathcal{P}}{A} \cup \mathcal{P}^{\mathcal{L}} = \frac{\mathcal{P}^N}{A}$. But this contradicts the fact that A is the minimal model of $\frac{\mathcal{P}^N}{A}$. □

There are some very important consequences of the above theorem. One we already established in Theorem 2: the number of stable models of a finite parametrized logic program is finite. The problem is that each of these stable models is infinite. This is precisely where Theorem 4 gives its fundamental contribution. It presents a finite representation of each of the stable models of $\mathcal{P}$.

Our aim now is to compute the finite representations of the stable models of $\mathcal{P}$. The implicit algorithm in the construction of $\mathcal{P}^N$ is quite basic. It just looks at all possible relations between formulas of the parameter logic appearing in the program. We now develop a more efficient implementation, assuming some mild conditions about the parameter logic. These allow us to prune some search paths in the construction of a normal logic program from $\mathcal{P}$.

Let $\mathcal{L} = \langle L, \vdash_{\mathcal{L}} \rangle$ be a monotonic logic satisfying the following conditions. The first condition, dubbed (Bot) is the existence of a bottom element in the language, i.e., $\bot \in L$ such that for any subset $\Phi \subseteq L$ we have that if $\Phi \vdash_{\mathcal{L}} \bot$ then $\Phi \vdash_{\mathcal{L}} \varphi$ for every $\varphi \in L$. This condition allows to detect an inconsistent set of formulas by checking if it entails $\bot$. The second condition, dubbed (Prop), is that L is built from a set of propositional symbols P and it satisfies: if $propSymb(\Phi) \cap propSymb(\varphi) = \emptyset$ and $\nvdash_{\mathcal{L}} \varphi$ then $\Phi \nvdash_{\mathcal{L}} \varphi$. Intuitively this condition imposes that if a non tautological formula does not have propositional symbols in common with a set of formulas, then it should not be entailed by that set of formulas.

Note that these two conditions are quite mild, and they are satisfied by every example of parameter logic we have shown above. For a parameter logic satisfying these conditions, we can sketch an algorithm, in Fig. 2, that, given a finite parametrized logic program $\mathcal{P}$, returns a normal logic program $\mathcal{P}^{alg}$. This algorithm is an improvement of the one constructing $\mathcal{P}^N$, by pruning several search paths using the conditions imposed on the parameter logic.

We can then prove that the pruned paths do not affect the result of the algorithm, i.e., the constructed program $\mathcal{P}^{alg}$ has the same stable models as $\mathcal{P}^N$. Given these improvements, the algorithm for constructing $\mathcal{P}^{alg}$ does not, in general, return exactly the normal logic program $\mathcal{P}^N$. In fact, one can readily

```
input: finite parametrized logic program 𝒫
 set i = 1; k = lenght(head(𝒫)); P^alg := P ∪ {φ ← : φ ∈ form(𝒫) and ⊢_ℒ φ}
  while i ≤ k
   for each subset A = {δ_1, ..., δ_i} of head(𝒫) of size i
     if A ⊢_ℒ ⊥ then /* A is inconsistent */
       for each φ ∈ form(𝒫) \ A
         add φ ← δ_1, ..., δ_i to 𝒫^alg unless
          there is φ ← ψ_1, ..., ψ_n ∈ 𝒫^alg with {ψ_1, ..., ψ_n} ⊆ A
     else
       for each φ ∈ form(𝒫) \ A such that propSymb(A) ∩ propSymb(φ) ≠ ∅
         if A ⊢_ℒ φ then
           add φ ← δ_1, ..., δ_i to 𝒫^alg unless
            there is φ ← ψ_1, ..., ψ_n ∈ 𝒫^alg with {ψ_1, ..., ψ_n} ⊆ A
   i=i+1
return 𝒫^alg
```

Fig. 2. Construction of $\mathcal{P}^{alg}$

see that $\mathcal{P}^{alg} \subseteq \mathcal{P}^N$. As expected, we can, nevertheless, prove that the extra rules of $\mathcal{P}^N$ are redundant, in the sense that the set of stable model of $\mathcal{P}^N$ and $\mathcal{P}^{alg}$ is the same.

Proposition 1. *Let $\mathcal{L}$ be a monotonic logic satisfying conditions (Bot) and (Prop). Then, for any finite parametrized logic program $\mathcal{P}$ over $\mathcal{L}$, we have that*

$$AS(\mathcal{P}^{alg}) = AS(\mathcal{P}^N).$$

Proof. It follows immediately from the constructions of $\mathcal{P}^{alg}$ and of $\mathcal{P}^N$ that $\mathcal{P}^{alg} \subseteq \mathcal{P}^N$. This implies that $Mod(\mathcal{P}^N) \subseteq Mod(\mathcal{P}^{alg})$. Moreover, given S a subset of $head(\mathcal{P})$, we can readily see that if a rule $r = \varphi \leftarrow \delta_1, \ldots, \delta_n$ is such that $r \in \frac{\mathcal{P}^N}{S}$ but $r \notin \frac{\mathcal{P}^{alg}}{S}$, then there exists $r' = \varphi \leftarrow \psi_1, \ldots, \psi_m \in \frac{\mathcal{P}^{alg}}{S}$ such that $\{\psi_1, \ldots, \psi_m\} \subset \{\delta_1, \ldots, \delta_n\}$. From this observation it follows that $Mod(\frac{\mathcal{P}^N}{S}) = Mod(\frac{\mathcal{P}^{alg}}{S})$. Therefore, S is a stable model of $\mathcal{P}^N$ (minimal model of $\frac{\mathcal{P}^N}{S}$) iff S is a stable model of $\mathcal{P}^{alg}$ (minimal model of $\frac{\mathcal{P}^{alg}}{S}$).

The above proposition is important since it allows the algorithm of $\mathcal{P}^{alg}$ to actually construct a finite representation of the stable models of a finite parametrized logic program $\mathcal{P}$. This can be done by constructing the normal logic program $\mathcal{P}^{alg}$ from $\mathcal{P}$ and then calculating the stable models of $\mathcal{P}^{alg}$. The latter can be done using any ASP solver. Note that, as we aimed, this construction uses in a modular way a reasoner for the parameter logic and reasoner for the stable model semantics. The reasoner for the parameter logic is only used for the construction of $\mathcal{P}^{alg}$. Then, an ASP solver can be used to obtain the stable models of $\mathcal{P}^{alg}$, which are the finite representations of the stable models of $\mathcal{P}$.

Regarding complexity, it should be clear that the use of parametrized logic programs, with default negation, increases the complexity of the parameter $\mathcal{L}$

alone. This comes from the fact that the stable model semantics, with default negation, adds, as usual, one extra level of non-determinism. From the point of view of logic programming there is also an exponential increasing in the complexity. Recall that in the construction of both $\mathcal{P}^N$ and $\mathcal{P}^{alg}$ we need to query an $\mathcal{L}$-oracle an exponential number of times. Moreover, we then need to compute the stable models of $\mathcal{P}^{alg}$ which, in the extreme case, can have exponentially more rules than the initial program $\mathcal{P}$.

This extra complexity is not surprising given the expressivity of the parametrized logic programs. Recall that a parametrized logic program can have any complex parametrized formula in the head and body of its rules. In some particular applications, however, there is no need for this general expressivity, and we can play the usual game between expressivity and complexity. We end this section with an example showing that we can consider restricted classes of parameter logics that have a more amenable complexity. These restricted languages may well have the necessary expressivity for modeling non-trivial scenarios.

An interesting example is the case of parametrized logic programs over a modal language that only contains literals, the necessity modal operator applied to literals and negations of the necessity operator applied to literals.

Note that we can capture the possibility operator $\Diamond$ since $\Diamond\ell \equiv_m \neg\Box\overline{\ell}$, where $\overline{\ell}$ is the complementary literal of ℓ, i.e., $\overline{\ell} = p$ if $\ell = \neg p$ and $\overline{\ell} = \neg p$ if $\ell = p$. In this restricted language the interaction between modal formulas is limited and, depending on which modal logic axioms the particular logic satisfies, we can construct the normal program $\mathcal{P}^{alg}$ from $\mathcal{P}$ in a simple way.

Proposition 2. *Let $\mathcal{P}$ be a finite parametrized logic program over a modal language only with literals, necessity applied to literals and negations of necessity applied to literals. Consider the following sets*

$$\begin{aligned}
\mathcal{P}_K =& \{\varphi \leftarrow \bot : \varphi \in form(\mathcal{P})\} \cup \\
& \{\bot \leftarrow p, \neg p : \{p, \neg p\} \subseteq head(\mathcal{P})\} \cup \\
& \{\bot \leftarrow \Box p, \Box\neg p : \{\Box p, \Box\neg p\} \subseteq head(\mathcal{P})\} \cup \\
& \{\bot \leftarrow \Box\ell, \neg\Box\ell : \{\Box\ell, \neg\Box\ell\} \subseteq head(\mathcal{P})\}. \\
\mathcal{P}_D =& \{\neg\Box\neg p \leftarrow \Box p : \Box p \in head(\mathcal{P}) \text{ and } \neg\Box\neg p \in form(\mathcal{P})\} \cup \\
& \{\neg\Box p \leftarrow \Box\neg p : \Box\neg p \in head(\mathcal{P}) \text{ and } \neg\Box p \in form(\mathcal{P})\}. \\
\mathcal{P}_T =& \{\ell \leftarrow \Box\ell : \Box\ell \in head(\mathcal{P}) \text{ and } \ell \in form(\mathcal{P})\}.
\end{aligned}$$

Then,

- *if $\mathcal{L}$ is the modal logic K then $\mathcal{P}^{alg} = \mathcal{P} \cup \mathcal{P}_K$;*
- *if $\mathcal{L}$ is the modal logic KD then $\mathcal{P}^{alg} = \mathcal{P} \cup \mathcal{P}_K \cup \mathcal{P}_D$;*
- *if $\mathcal{L}$ is the modal logic KT then $\mathcal{P}^{alg} = \mathcal{P} \cup \mathcal{P}_K \cup \mathcal{P}_T$;*
- *if $\mathcal{L}$ is the modal logic KTD then $\mathcal{P}^{alg} = \mathcal{P} \cup \mathcal{P}_K \cup \mathcal{P}_T \cup \mathcal{P}_D$.*

Proof. The result follows easily from the observation that, for this restricted language, we have that $\Phi \vdash_K \varphi$ iff one of the following conditions holds: for

some propositional symbol p, $\{p, \neg p\} \subseteq \Phi$ or $\{\Box p, \Box \neg p\} \subseteq \Phi$; or $\{\Box \ell, \neg \Box \ell\} \subseteq \Phi$ for some literal ℓ. If we add to K the seriality axiom D then we can also entail φ from a set Φ of formulas if $\Box p \in \Phi$ and $\varphi = \neg \Box \neg p$, or when $\Box \neg p \in \Phi$ and $\varphi = \neg \Box p$. In the case of the addition of the transitivity axiom T we can also conclude φ from a set Φ of formulas if $\Box \varphi \in \Phi$.

The above proposition is important because it gives a way to construct $\mathcal{P}^{alg}$ using only syntactical checks, i.e., we do not need to use a modal logic oracle. This is only possible because the interaction between modal formulas in this restricted language is limited and can be clearly described using the above rules. The four rules of $\mathcal{P}_K$ are related to contradictions. The first one refers to the so-called explosion principle: from a contradiction everything follows. The others express how to detect an inconsistency. The rules of $\mathcal{P}_D$ are related to the connection between necessity and possibility: if something is necessary then it is possible, which holds in a modal logic satisfying D. The need for the rules in $\mathcal{P}_T$ comes from the fact that a formula follows from its necessity in a modal logic satisfying the reflexivity axiom T. For lack of space, we did not add several more examples of modal logics to the above proposition. Just to give an example, in the case of doxastic modal logic, which is usually assumed to be a $KD45$ modal logic, we have that $\mathcal{P}^{alg} = \mathcal{P} \cup \mathcal{P}_K \cup \mathcal{P}_D$.

Regarding complexity, it is interesting to note that the maximum number of rules added to $\mathcal{P}^{alg}$ is linear in the number of rules of $\mathcal{P}$.

4 Conclusions and future work

In this paper we have proved decidability for parametrized logic programs, assuming the decidability of the parameter logic. We have provided an implementation that combines in modular way a reasoner for a decidable parameter logic with an answer set solver. We have studied examples of modal logics in a restricted language. For those, the construction of a normal logic program $\mathcal{P}^{alg}$ from a given parametrized logic program $\mathcal{P}$ does not need to use a modal logic oracle, and, moreover, the number of rules added to $\mathcal{P}$ in order to obtain $\mathcal{P}^{alg}$ is at most linear in the number of rules of $\mathcal{P}$.

Regarding future work, we want to implement the algorithms presented in this paper in the case of interesting parameter logics. One such example is the case of standard deontic logic, which would then allow us to construct a declarative non-monotonic framework for specifying normative systems [3]. We also want to study in more detail the natural connection between parametrized logic programming and the general approach of multi-context systems [4], along the lines of [12]. Another interesting topic is to investigate belief change in our setting, which would be a challenging problem due to the known difficulties in combining belief change of rules and belief change in classical logic [17], although recent developments have shown a possible unifying view [18, 19]. Finally, we would like to study the well-founded semantics for parametrized logic programs along the lines of what is done in [14] for hybrid MKNF.

References

1. A. Artosi, P. Cattabriga, and G. Governatori. Ked: A deontic theorem prover. In *Workshop on Legal Application of Logic Programming*, pages 60–76. IDG, 1994.
2. N. Belnap. A useful four-valued logic. In G. Epstein and M. Dunn, editors, *Modern Uses of Multiple-Valued Logic*, pages 7–37. Reidel Publishing Company, 1977.
3. G. Boella, L. van der Torre, and H. Verhagen. Introduction to the special issue on normative multiagent systems. *JAAMAS*, 17(1):1–10, 2008.
4. G. Brewka and T. Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *AAAI*, pages 385–390. AAAI Press, 2007.
5. B. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.
6. P. Ferraris. Logic programs with propositional connectives and aggregates. *ACM Trans. Comput. Log.*, 12(4):25, 2011.
7. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *Clasp* : A conflict-driven answer set solver. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *LPNMR*, volume 4483 of *LNCS*, pages 260–265. Springer, 2007.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.
9. R. Gonçalves and J. J. Alferes. Parametrized logic programming. In T. Janhunen and I. Niemelä, editors, *JELIA*, volume 6341 of *LNCS*, pages 182–194. Springer, 2010.
10. R. Gonçalves and J. J. Alferes. An embedding of input-output logic in deontic logic programs. In T. Ågotnes, J. Broersen, and D. Elgesem, editors, *DEON*, volume 7393 of *LNCS*, pages 61–75. Springer, 2012.
11. R. Gonçalves and J. J. Alferes. Specifying and reasoning about normative systems in deontic logic programming. In W. van der Hoek, L. Padgham, V. Conitzer, and M. Winikoff, editors, *AAMAS*, pages 1423–1424. IFAAMAS, 2012.
12. M. Homola, M. Knorr, J. Leite, and M. Slota. MKNF knowledge bases in multi-context systems. In M. Fisher, L. van der Torre, M. Dastani, and G. Governatori, editors, *CLIMA*, volume 7486 of *LNCS*, pages 146–162. Springer, 2012.
13. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In *KR*, pages 57–67. AAAI Press, 2006.
14. M. Knorr, J. J. Alferes, and P. Hitzler. Local closed world reasoning with description logics under the well-founded semantics. *Artif. Intell.*, 175(9–10):1528–1554, 2011.
15. B. Motik and R. Rosati. Reconciling description logics and rules. *JACM*, 57(5), 2010.
16. B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.
17. M. Slota and J. Leite. On semantic update operators for answer-set programs. In H. Coelho, R. Studer, and M. Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 957–962. IOS Press, 2010.
18. M. Slota and J. Leite. A unifying perspective on knowledge updates. In L. F. del Cerro, A. Herzig, and J. Mengin, editors, *JELIA*, volume 7519 of *LNCS*, pages 372–384. Springer, 2012.
19. M. Slota and J. Leite. The rise and fall of semantic rule updates based on SE-models. *Theory and Practice of Logic Programming (TPLP)*, 2013. To appear.
20. G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.
21. R. Wójcicki. *Theory of Logical Calculi*. Synthese Library. Kluwer, 1988.