Combining rules and ontologies via parametrized logic programs

Ricardo Gonçalves
NOVA LINCS
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 explore the use of description logics as parameter logics, and show the expressivity of this framework for combining rules and ontologies.

Introduction

Parametrized logic programming (Gonçalves and Alferes 2010) was introduced as an extension of answer set programming (Gelfond and Lifschitz 1988) 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, being the only difference 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, (Gonçalves and Alferes 2010) 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 (Motik and Rosati 2010), using an appropriate choice of the parameter logic. In (Gonçalves and Alferes 2012) deontic logic programs are introduced using standard deontic logic (von Wright 1951) as the parameter logic. Moreover, in (Gonçalves and Alferes 2013) the decidability and implementation of parametrized logic was discussed.

Parametrized logic programming can thus be seen as a framework which allows 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 of the monotonic logic at hand.

In recent years, there has been a considerable amount of effort devoted to combining Description Logics (DLs) with logic programming non-monotonic rules – see, e.g., related work in (Eiter et al. 2008; Motik and Rosati 2010).

In this paper we explore precisely the use of description logics as parameter logics, and show the expressivity of the resulting framework for combining rules and ontologies.

Parametrized logic programs

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 section we introduce the syntax and semantics of normal parametrized logic programs (Gonçalves and Alferes 2010).

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 \( \mathcal{L} = \langle \mathcal{L}, \vdash_\mathcal{L} \rangle \) where \( \mathcal{L} \) is a set of formulas and \( \vdash_\mathcal{L} \) is a Tarskian consequence relation (Wójcicki 1988) over \( \mathcal{L} \), i.e., satisfying the following conditions, for every \( T \cup \Phi \subseteq \mathcal{L} \),

\[
\begin{align*}
\text{Reflexivity: } & \text{ if } \varphi \in T \text{ then } T \vdash_\mathcal{L} \varphi; \\
\text{Cut: } & \text{ if } T \vdash_\mathcal{L} \varphi \text{ for all } \varphi \in \Phi, \text{ and } \Phi \vdash_\mathcal{L} \psi \text{ then } T \vdash_\mathcal{L} \psi; \\
\text{Weakening: } & \text{ if } T \vdash_\mathcal{L} \varphi \text{ and } T \subseteq \Phi \text{ then } \Phi \vdash_\mathcal{L} \varphi. 
\end{align*}
\]

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 \( \mathcal{L} \) closed under the relation \( \vdash \). One fundamental characteristic of the above definition of monotonic logic is that it has as a consequence that, for every (monotonic) logic
the above mentioned interdependence.

ories (sets of formulas closed under the consequence of the

parametrized atoms are not independent of each other.

tations. The problem is that, contrary to the case of atoms,
to take sets of formulas of the parameter logic as interpre-
tute atoms by formulas of a parameter logic, the first idea is
t of atoms. In a parametrized logic program, since we substi-

Given this general language of parametrized logic programs,

\[ L \]

\[ \{ \text{head}(r) = \varphi, \text{body}^+(r) = \{ \psi_1, \ldots, \psi_n \}, \text{body}^-(r) = \{ \delta_1, \ldots, \delta_m \} \} \]

and

\[ \text{body}^+(r) = \text{body}^+(r) \cup \text{body}^-(r). \]

A parametrized logic program \( P \) we define \( \text{form}(P) \) to be the set of all formulas of the parameter language \( L \) appearing in \( P \), i.e.,

\[ \text{form}(P) = \bigcup_{r \in P} (\{ \text{head}(r) \} \cup \text{body}(r)). \]

We also define the set

\[ \text{head}(P) = \{ \text{head}(r) : r \in P \}. \]

**Semantics**

Given this general language of parametrized logic programs,

we define its stable model semantics, as generalization of

the stable model semantics (Gelfond and Lifschitz 1988) of

normal logic programs.

In the traditional approach an interpretation is just a set

of atoms. In a parametrized logic program, since we substi-
tute atoms by formulas of a parameter logic, the first idea is
to take sets of formulas of the parameter logic as interpre-
tations. 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 rela-
tion of the parameter logic. For example, if we take classical
propositional logic (CPL) as the parameter logic, we have
that if the parametrized atom \( p \land q \) is true then so are the
parametrized atoms \( p \) and \( q \). If we take, for example, stan-
dard deontic logic SDL (von Wright 1951) as parameter, we
have that, since \( O(p \lor q), O(\neg p) \vdash_{SDL} O(q), \) any SDL logi-
cal theory containing both \( O(p \lor q) \) and \( O(\neg p) \) also contains
\( O(q) \).

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

**Definition 2** A normal \( L \) parametrized logic program is a set of rules

\[ \varphi \leftarrow \psi_1, \ldots, \psi_n, \text{not} \delta_1, \ldots, \text{not} \delta_m \quad (1) \]

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

A definite \( L \) parametrized logic program is a set of rules without negations as failure, i.e., of the form

\[ \varphi \leftarrow \psi_1, \ldots, \psi_n \text{ where } \varphi, \psi_1, \ldots, \psi_n \in L. \]

As usual, the symbol \( \leftarrow \) represents rule implication, the
symbol \( \lor \) represents conjunction and the symbol \( \text{not} \)
represents default negation. A rule as (1) has the usual
reading that \( \varphi \) 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

\[ \text{head}(r) = \varphi, \text{body}^+(r) = \{ \psi_1, \ldots, \psi_n \}, \text{body}^-(r) = \{ \delta_1, \ldots, \delta_m \} \]

and

\[ \text{body}^+(r) = \text{body}^+(r) \cup \text{body}^-(r). \]

An interpretation is a model of logic program \( P \) if it satisfies every rule of \( P \). We denote by \( \text{Mod}_L(P) \) the set of models of \( P \).

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 \( L \) parametrized
logic program.

**Theorem 1** Every definite \( L \) parametrized logic program has a least model.

We denote by \( S_L^* \) the least model of a definite program \( P \).

It is important to note that Theorem 1 holds for every
choice of the parameter logic \( L \).

The stable model semantics of a normal \( L \) parametrized
logic program is defined using a Gelfond-Lifschitz like op-
erator.

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

- remove from \( P \) all rules which contain a literal not \( \varphi \) such that \( T \models_L \varphi; \)
- remove from the remaining rules all default literals.

Since \( P_{\Gamma} \) is a definite \( L \) 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 \( \Gamma \) operator.

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

An important feature of parametrized logic programming
is that its stable model semantics is independent of the
semantics of the parameter logic, since the central concept is
the consequence relation of the parameter logic.

Let us now show an example of how parametrized logic
programs can be used to combine a monotonic formalism
with a non-monotonic one. We choose three different logics
over the same propositional language.
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 ($\neg, \lor, \land, \Rightarrow$). Many consequence relations can be defined over this language. We present three interesting examples: classical logic, Belnap’s paraconsistent logic and intuitionistic logic. Consider the following programs:

- $P_1 \{ \begin{align*}
p & \leftarrow \neg q \\
p & \leftarrow q
\end{align*} \}$
- $P_2 \{ \begin{align*}
p & \leftarrow \neg q \lor q
\end{align*} \}$
- $P_3 \{ \begin{align*}
q & \leftarrow (q \lor s) \Rightarrow p \\
r & \leftarrow \neg p \\
r & \leftarrow (p \lor q) \Rightarrow r \\
s & \leftarrow q
\end{align*} \}$
- $P_4 \{ \begin{align*}
p & \leftarrow \neg q, \neg r \\
p & \leftarrow q \lor q
\end{align*} \}$
- $P_5 \{ \begin{align*}
p & \leftarrow \neg q, \neg r
\end{align*} \}$
- $P_6 \{ \begin{align*}
p & \leftarrow (q \lor q)
\end{align*} \}$

Let $\mathcal{L} = \langle L, \models_{\text{CPL}} \rangle$ be Classical Propositional Logic (CPL) over the language $L$. Let us study the semantics of $P_1$. Note that every logical theory of CPL that does not contain neither $p$ nor $\neg p$ satisfies $P_1$. In particular, the set $\text{Taut}$ of tautologies of CPL is a model of $P_1$. So, $\text{Taut} \models_{\text{CPL}} = \text{Taut}$.

This means that $p, \neg p, q, \neg q \notin S^p_{\text{CPL}}$. We also have that $S^p_{\text{CPL}} = \{ p \}$. So, in the case of $P_2$ we have that $p \in S^p_{\text{CPL}}$. Also, we have that $r \in S^r_{\text{CPL}}$ and $s \in S^s_{\text{CPL}}$.

In the case of $P_3$, its stable models are the theories of CPL that contain $p$ and do not contain $q$ and $\neg q$. Therefore, we can conclude that $p \in S^p_{\text{CPL}}$. In the case of $P_6$, since $(p \lor \neg p) \in T$ for every logical theory $T$ of CPL we can conclude that the only stable model of $P_6$ is the set $\text{Theo}$ of theorems of CPL. Therefore $p \notin S^p_{\text{CPL}}$.

Consider now $\mathcal{L} = \langle L, |=_{\text{IFL}} \rangle$ 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 $\neg p, (p \lor q) \models q$. Therefore we have that $q, s \notin S^p_{\text{Four}}$.

Let now $\mathcal{L} = \langle L, |=_{\text{IPL}} \rangle$ be the propositional intuitionistic logic IPL. It is well-known that $q \lor \neg q$ is not a theorem of IPL. Therefore, considering program $P_2$, we have $S^p_{\text{IPL}} = \emptyset$. So, contrarily to the case of CPL, we have that $p \notin S^p_{\text{IPL}}$.

Combining rules and ontologies

In this section we discuss the use of description logics as parameter logic in the framework of parametrized logic programming. We will then illustrate the expressivity of the resulting framework to combine non-monotonic rules and ontologies.

In what follows, and for simplicity, we use description logic $\mathcal{ALC}$ (Schmidt-Schauß and Smolka 1991). We start by briefly recalling the syntax and semantics of $\mathcal{ALC}$. For a more general and thorough introduction to DLs we refer to (Baader et al. 2010). The language of $\mathcal{ALC}$ is defined over countably infinite sets of concept names $\mathcal{NC}$, role names $\mathcal{NR}$, and individual names $\mathcal{NI}$ as shown in the upper part of Table 1. Building on these, complex concepts are introduced in the middle part of Table 1, which, together with atomic concepts, form the set of concepts. We conveniently denote individuals by $a$ and $b$, (atomic) roles by $R$ and $S$, atomic concepts by $A$ and $B$, and concepts by $C$ and $D$. All expressions in the lower part of Table 1 are axioms. A concept equivalence $\equiv$ is an abbreviation for $C \equiv D$ and $D \equiv C$. Concept and role assertions are $\text{ABox axioms}$ and all other axioms $\text{TBox axioms}$, and an ontology is a finite set of axioms.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic concept</td>
<td>$A \in \mathcal{NC}$</td>
<td>$\Delta^I \subseteq \Delta^T$</td>
</tr>
<tr>
<td>Atomic role</td>
<td>$R \in \mathcal{NR}$</td>
<td>$R^I \subseteq \Delta^T \times \Delta^T$</td>
</tr>
<tr>
<td>Individual</td>
<td>$a \in \mathcal{NI}$</td>
<td>$a^I \in \Delta^I$</td>
</tr>
</tbody>
</table>

The semantics of $\mathcal{ALC}$ is defined in terms of interpretations $\mathcal{I} = (\Delta^I, \Gamma)$, which consist of a non-empty domain $\Delta^I$ and an interpretation function $\Gamma$. The latter is defined for (arbitrary) concepts, roles, and individuals as in Table 1. Moreover, an interpretation $\mathcal{I}$ satisfies an axiom $\alpha$, written $\mathcal{I} \models \alpha$, if the corresponding condition in Table 1 holds. If $\mathcal{I}$ satisfies all axioms in an ontology $\mathcal{O}$, then $\mathcal{I}$ is a model of $\mathcal{O}$, written $\mathcal{I} \models \mathcal{O}$. If $\mathcal{O}$ has at least one model, then it is called consistent, otherwise inconsistent. Also, $\mathcal{O}$ entails axiom $\alpha$, written $\mathcal{O} \models \alpha$, if every model of $\mathcal{O}$ satisfies $\alpha$.

Given the consequence relation of $\mathcal{ALC}$ we can now illustrate how $\mathcal{ALC}$ can be used as parameter logic.

Example 2 The following program $(P_1)$ is an adaptation of an example taken from (Motik and Rosati 2007), which uses MKNF knowledge bases to combine rules and ontologies. The scenario is about determining the car insurance pre-
mium based on various information about the driver.

NotMarried ≡ ¬Married ←
NotMarried ⊑ HighRisk ←

∃Spouse. T ⊑ Married ←
NotMarried(x) ← p(x), not Married(x)
Discount(x) ← Spouse(x, y), p(x), p(y)
p(Jonh) ←

Note that in parametrized logic programming the combination of an ontology with a rule system can be done in a natural way, simply by adding the ontology elements as facts of the rule system. As usual in logic programming, variables in rules stand for all their possible instantiations by individuals appearing in the program.

Program P3 can be rewritten in order to remove its first rule, which is nothing but an artificial tool to overcome the impossibility of having complex DL formulas in the head of MKNF rules (in this case, having the classical negation of an atom in a head). Moreover, we may also add bodies to the facts coming from the ontology. E.g. we can add a non-exceptional period, obtaining exceptional periods, obtaining P2

Consider now program P3 obtained by adding to P2 the facts: Spouse(Bob, Ann), p(p) ←, and p(Ann) ←. Every stable model of P3 contains Discount(Bob), and so it entails Discount(Bob).

Conclusions

In this paper we have discussed the use of the framework of parametrized logic programming for combining non-monotonic rules and ontologies. This approach is quite expressive since it allows complex DL axioms to appear both in the body and in the head of non-monotonic rules.

In (Gonçalves and Alferes 2010) the authors show how parametrized logic programming can capture the semantics of MKNF hybrid knowledge bases (Motik and Rosati 2010) by an appropriate choice of the parameter logic. As future work we aim to study the relation between parametrized logic programs and other frameworks for combining rules and ontologies, e.g., the DL-programms of (Eiter et al. 2008).

Acknowledgments

Ricardo Gonçalves was supported by FCT under project ERRO (PTDC/EIA-CCO/121823/2010).

References


