∃-ASP

Fabien Garreau, Laurent Garcia, Claire Lefèvre and Igor Stéphan
LERIA
University of Angers
{fgarreau,garcia,claire,stephan}@info.univ-angers.fr

Abstract

Answer set programming (ASP) is an appropriate formalism to represent various problems issued from artificial intelligence and arising when available information is incomplete. When dealing with information expressed in terms of ontologies in some tractable description logic language, ASP must be extended to handle existential variables. We present the syntax and semantics of an ASP language with existential variables using Skolemization. We formalize its links with standard ASP. This work has led to an implementation.

Introduction

This paper deals with the treatment of ontologies in Answer Set Programming (ASP) (Gelfond and Lifschitz 1988). We are interested in using ASP technologies for querying large scale multisource heterogeneous web information. ASP is considered to handle, by using default negation, inconsistencies emerging by the fusion of the sources expressed by scalable description logics. Moreover, ASP can enrich the language of ontologies by allowing the expression of default information (for instance, when expressing inclusion of concepts with exceptions). When dealing with ontologies in ASP, the problem stems from the presence of existential variables in description logics which are not expressible in normal logic programs. The present work proposes a definition of ASP with existential variables in order to express, in a unique formalism, ontologies enriched by default negation and rules. Processing existential variables is done in terms of Skolemization.

The study of the combination of ontologies and rules is not new (Rosati 2006; Eiter et al. 2008; de Bruijn et al. 2010; Motik and Rosati 2010; Ferraris, Lee, and Lifschitz 2011; Lee and Palla 2011; Magka, Krötzsch, and Horrocks 2013). In most of these models, the knowledge base is viewed as a hybrid knowledge base composed of two parts ($\mathcal{T}$,$\mathcal{P}$): $\mathcal{T}$ is a knowledge base describing the ontological information expressed with a fragment of first-order logic, using for example description logic, and $\mathcal{P}$ describes the rules in terms of a logic program.

The miscellaneous attempts to integrate the two formalisms can be distributed into three classes (Eiter et al. 2008; Lee and Palla 2011).

In the first class (like in (Eiter et al. 2008)), the two formalisms are handled separately. $\mathcal{T}$ is seen as an external source of information which can be used by the logic program through special predicates querying the DL base. The two bases are then independent with their own semantics and the link between the two bases is performed using these special predicates. (Eiter et al. 2013) uses their extension of ASP with external atoms to simulate rules with existential variables in the head (external atoms in the body serve to introduce new null values).

The second case (like in (Rosati 2006; Motik and Rosati 2010)) corresponds to an hybrid formalism which integrates DLs and rules in a consistent semantic framework. Predicates of $\mathcal{T}$ can be used in the rules of the program. Nevertheless, there are some restrictions: for instance, these predicates can not be used in the negative part of the body of a rule.

The last case integrates DLs and rules in a unique formalism. For instance, (de Bruijn et al. 2010) uses quantified equilibrium logic (QEL). In this work, several hybrid knowledge bases are defined (with *safe restriction*, *safe restriction without unique name assumption* or with *guarded restriction*) and it is proved that each category and their models can be expressed in terms of QEL.

A large part of these works concerns the questions of complexity and decidability. In these frameworks, existential variables are allowed in the part of the ontological information but are not allowed in the head of the rules.

Next to these models, (Ferraris, Lee, and Lifschitz 2011) proposes a model allowing to cover both stable models semantics and first-order logic by means of a second-order formula issued from the initial information. Its links with the previously cited works have been established in (Lee and Palla 2011). (You, Zhang, and Zhang 2013) proposes an extension of ASP with existential variables in rule heads whose semantics corresponds to that of (Ferraris, Lee, and Lifschitz 2011).

Other works in logic programming take their origin in Datalog and extend the language for specifying ontologies. Datalog+/- is a family of such extensions with syntactical restrictions so that decidability is ensured. Several approaches with existential quantified variables based on Datalog+/- have been proposed in the literature but some have no non monotonic negation (Alviano et al. 2012) and other have

only stratified negation (Cali et al. 2010). Nevertheless, one important and interesting point of these works is that they focus on queries which is an important issue when dealing with ontologies.

In (Magka, Krötzsch, and Horrocks 2013), the knowledge base is a single one allowing existential variables and default negation in the same rule. This work studies some conditions of acyclicity and stratification that must be verified by the base ensuring the existence of a unique finite stable model. The work is both theoretical and practical but it is concerned with a limited extension of ASP.

As far as we know, the only works leading to an implementation are those of (Ianni et al. 2005; Eiter et al. 2005), based on (Eiter et al. 2008), and of (Magka, Krötzsch, and Horrocks 2013) which has been applied to information about biochemistry.

The aim of our present work is to describe knowledge in a single framework which can lead to useful implementation. We focus on ASP because it is a powerful framework for knowledge representation and provides efficient solvers. The work consists in enriching the ASP framework to take into account existential variables. It can be seen as the other side of the work consisting in introducing nonmonotonicity in existential rules (Baget et al. 2014b; 2014a).

Next section gives the preliminary notions and definitions useful for the paper. Then, we define programs expressed in $\exists$-ASP, an extension of ASP allowing existential variables, and answer sets on this kind of programs. Last, we give the links between $\exists$-ASP and standard ASP with a method to translate a program expressed in $\exists$-ASP into a program expressed in (standard) ASP and proofs about the transformation.

Preliminaries

In this section, we give the formal definitions of the language and some notions useful in the following of the paper.

The set $\mathcal{V}$ denotes the infinite countable set of *variables*. A *language* $\mathcal{L}$ is defined as a triplet $(\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ which denotes respectively the set of *constant symbols*, the set of *function symbols* and the set of *predicate symbols* of the language. It is assumed that the sets $\mathcal{V}$, $\mathcal{CS}$, $\mathcal{FS}$ and $\mathcal{PS}$ of any language are disjoint. Function ar denotes the arity function from $\mathcal{FS}$ to $\mathbb{N}^*$ and from $\mathcal{PS}$ to $\mathbb{N}$ which associates to each function or predicate symbol its arity.

The set $\mathbf{T}(\mathcal{L})$ denotes the set of *terms* of a language $\mathcal{L} = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ defined by induction as follows:

- if $v \in \mathcal{V}$ then $v \in \mathbf{T}(\mathcal{L})$,
- if $c \in \mathcal{CS}$ then $c \in \mathbf{T}(\mathcal{L})$,
- if $f \in \mathcal{FS}$ with $ar(f) = n > 0$ and $t_1, \ldots, t_n \in \mathbf{T}(\mathcal{L})$ then $f(t_1, \ldots, t_n) \in \mathbf{T}(\mathcal{L})$.

The set $\mathbf{GT}(\mathcal{L})$ denotes the set of *ground terms* of a language $\mathcal{L} = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ defined by induction as follows:

- if $c \in \mathcal{CS}$ then $c \in \mathbf{GT}(\mathcal{L})$,
- if $f \in \mathcal{FS}$ with $ar(f) = n > 0$ and $t_1, \ldots, t_n \in \mathbf{GT}(\mathcal{L})$ then $f(t_1, \ldots, t_n) \in \mathbf{GT}(\mathcal{L})$.

The set $\mathbf{A}(\mathcal{L})$ denotes the set of *atoms* of a language $\mathcal{L} = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ defined as follows:

- if $a \in \mathcal{PS}$ with $ar(a) = 0$ then $a \in \mathbf{A}(\mathcal{L})$,
- if $p \in \mathcal{PS}$ with $ar(p) = n > 0$ and $t_1, \ldots, t_n \in \mathbf{T}(\mathcal{L})$ then $p(t_1, \ldots, t_n) \in \mathbf{A}(\mathcal{L})$.

The set $\mathbf{GA}(\mathcal{L})$ denotes the set of *ground atoms* of a language $\mathcal{L} = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ defined as follows:

- if $a \in \mathcal{PS}$ with $ar(a) = 0$ then $a \in \mathbf{GA}(\mathcal{L})$,
- if $p \in \mathcal{PS}$ with $ar(p) = n > 0$ and $t_1, \ldots, t_n \in \mathbf{GT}(\mathcal{L})$ then $p(t_1, \ldots, t_n) \in \mathbf{GA}(\mathcal{L})$.

A *substitution* over a language $\mathcal{L}$ is a mapping from the set of variables to the set of the terms $\mathbf{T}(\mathcal{L})$. Let t be a term (resp. a an atom) and σ a substitution, $\sigma(t)$ (resp. $\sigma(a)$) is an *instance* of t (resp. a).

A *ground substitution* over a language $\mathcal{L}$ is a mapping from the set of variables to the set of the ground terms $\mathbf{GT}(\mathcal{L})$. Let t be a term (resp. a an atom) and σ a ground substitution, $\sigma(t)$ (resp. $\sigma(a)$) is a *ground instance* of t (resp. a).

A *partial ground substitution* for a set of variables V over a language $\mathcal{L}$ is a mapping from V to the set of ground terms $\mathbf{GT}(\mathcal{L})$. Let t be a term (resp. a an atom) and σ a partial ground substitution for a set of variables V, $\sigma(t)$ (resp. $\sigma(a)$) is a *partial ground instance* of t (resp. a) w.r.t. the set of variables V.

Syntax and semantics of $\exists$-ASP

In this section, we define a variant of ASP allowing the use of existentially quantified variables (called *existential variables* in the sequel). The rules proposed here extend classical safe rules (without disjunction) of the form:

$$H \leftarrow B_1, \ldots, B_m, not\ N_1, \ldots, not\ N_s.$$

where $H, B_1, \ldots, B_m, N_1, \ldots, N_s$ are atoms. Safety imposes that all variables that appear in a rule also appear in the positive part of its body. In such a rule, all variables are interpreted as universally quantified. In the sequel, universally quantified variables will be called *universal variables*.

These classical rules are extended in two ways. First, the head of the rule, atom H, is replaced by a conjunction of atoms and each negated atom N_i is also replaced by a conjunction of atoms. These conjunctions allow multiple atoms to refer to the same existential variable. Second, the safety condition is relaxed by allowing these new conjunctions of atoms to contain variables that do not appear in the positive part of the rule. These variables are interpreted as existential ones.

For example, in the rule $(p(X, Y) \leftarrow q(X), not\ r(X, Z).)$, variable X is interpreted as universal, and Y and Z are interpreted as existential. The rule can be read as: "for all X, if $q(X)$ is true and there does not exist Z such that $r(X, Z)$ is true, then one can conclude that there exists Y such that $p(X, Y)$ is true".

Definition 1 ($\exists$-rule and $\exists$-program) *An $\exists$-program P of language $\mathcal{L} = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ is a set of $\exists$-rules r defined as follows ($m, s \geq 0$, $n, u_1, \ldots, u_s \geq 1$):*

$$H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m, not\ (N_1^1, \ldots, N_{u_1}^1), \ldots, not\ (N_1^s, \ldots, N_{u_s}^s).$$

with $H_1, \ldots, H_n, B_1, \ldots, B_m, N^1_1, \ldots, N^1_{u_1}, \ldots, N^s_1, \ldots, N^s_{u_s} \in \mathbf{A}(\mathcal{L})$.

We use the following notations:

- $head(r) = \{H_1, \ldots, H_n\}$.
- $body^+(r) = \{B_1, \ldots, B_m\}$.
- $body^-(r) = \{\{N^1_1, \ldots, N^1_{u_1}\}, \ldots, \{N^s_1, \ldots, N^s_{u_s}\}\}$.
- $\mathcal{V}(r)$ *the variables,*
- $\mathcal{V}_{H\exists}(r)$ *the variables which are in* $H_1, \ldots, H_n$ *but which are not in* $B_1, \ldots, B_m$ *(i.e. existential variables of the head of* r*),*
- $\mathcal{V}_\exists(r)(N^i_1, \ldots, N^i_{u_i})$ *variables which are in* $N^i_1, \ldots, N^i_{u_i}$ *but not in* $B_1, \ldots, B_m$, $1 \leq i \leq s$ *(i.e. existential variables of* $N^i_1, \ldots, N^i_{u_i}$*).*
- $\mathcal{V}_{N\exists}(r) = \bigcup_{1 \leq i \leq s} \mathcal{V}_\exists(r)(N^i_1, \ldots, N^i_{u_i})$,
- $\overline{\mathcal{V}_{N\exists}}(r) = \mathcal{V}(r) \setminus \mathcal{V}_{N\exists}(r)$,
- $\mathcal{V}_\exists(r) = \mathcal{V}_{H\exists}(r) \bigcup \mathcal{V}_{N\exists}(r)$
- $\mathcal{V}_{H\forall}(r)$ *the variables which are at least in* $H_1, \ldots, H_n$ *and in* $B_1, \ldots, B_m$ *(i.e. universal variables of the head of* r*, the frontier variables).*
- $\mathcal{V}_\forall(r)(N^i_1, \ldots, N^i_{u_i})$ *the variables which are at least in* $N^i_1, \ldots, N^i_{u_i}$ *and in* $B_1, \ldots, B_m$ *(i.e. universal variables of* $N^i_1, \ldots, N^i_{u_i}$*).*

Moreover, the sets $\mathcal{V}_\exists(r)(N^i_1, \ldots, N^i_{u_i})$ *for every* $1 \leq i \leq s$ *must be disjoint and the sets* $\mathcal{V}_{H\exists}(r)$ *and* $\mathcal{V}_{N\exists}(r)$ *must also be disjoint. (If a variable appears in several of the* $N^i_1, \ldots, N^i_{u_i}$ *or if it appears in* $H_1, \ldots, H_n$ *and in one of the* $N^i_1, \ldots, N^i_{u_i}$, $1 \leq i \leq s$*, then it must appear in* $B_1, \ldots, B_m$ *and it is a universal variable.)*

For all rules r *of a program* P, $\mathcal{V}_\exists(r)$ *must be disjoint (i.e. all the names of the existential variables of the program are different).*

A rule r *is a* definite rule *if* $body^-(r) = \emptyset$ *and a program is a* definite program *if all the rules are definite.*

Let us note that in such a rule r, several atoms are allowed in $head(r)$ and in each set of $body^-(r)$. In this case, a list of atoms must be seen as the conjunction of each atom of the list.

Concerning the variables involved in the rule, they can be quantified universally or existentially. The quantifiers are not explicitly expressed in the rule but they depend on the position in the rule: the variables appearing in $body^+(r)$ are universally quantified while the ones not appearing in $body^+(r)$ are existentially quantified. Let us note that the existential variables, in the head or in each negative part of the body, are locally quantified.

Example 1 *Let* P_U *be an* $\exists$*-program of language* $\mathcal{L}_U = (\{a\}, \emptyset, \{p, phdS, d, l, gC\})$ *with* $ar(p) = ar(d) = ar(l) = 1$ *and* $ar(phdS) = ar(gC) = 2$*. p stands for person, phdS for phDStudent, d for director, l for lecturer and gC for givesCourses.*

$$
\begin{aligned}
P_U = \{ & \\
& r_0 : p(a)., \\
& r_1 : l(a)., \\
& r_2 : phdS(X, D), d(D) \leftarrow \\
& \quad p(X), not(l(X), gC(X, Y)).\}
\end{aligned}
$$

The rule r_2 *means that for a person X there exists a director D and X is a phD student of D, unless X is a lecturer and it exists a course given by X.*

We have $\mathcal{V}_{H\forall}(r) = \{X\}$, $\mathcal{V}_{H\exists}(r) = \{D\}$, $\mathcal{V}_\exists(r)(l(X), gC(X, Y)) = \{Y\}$, $\overline{\mathcal{V}_{N\exists}}(r) = \{X, D\}$.

For each program P, we consider that its language $\mathcal{L}_P = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ consists of exactly the constant symbols, function symbols and predicate symbols appearing in P.

Proposition 1 *Any (first-order classical) ASP program is an* $\exists$*-program.*

Proof 1 *This is a direct consequence of Definition 1.*

The semantics of $\exists$-programs uses Skolemization of existential variables appearing in the heads of the rules. We now define this Skolemization.

Definition 2 (Skolem symbols) *Let* r *be an* $\exists$*-rule,* n *the cardinality of* $\mathcal{V}_{H\forall}(r)$ *and* $Y \in \mathcal{V}_{H\exists}(r)$ *an existential variable of* r *then* sk^n_Y *is a Skolem function symbol of arity* n *(if* $n = 0$ *then* sk_Y *is a Skolem constant symbol).*

Example 2 (Example 1 continued) *Symbol* sk^1_D *is a Skolem function symbol of arity 1 for the existential variable D of the head of the rule* r_2*.*

Definition 3 (Skolem Program) *Let* P *be an* $\exists$*-program of language* $\mathcal{L}_P$*.*

Let s *be an ordered sequence of the variables* $\mathcal{V}_{H\forall}(r)$ *of an* $\exists$*-rule* r *of* P*. sk(r) denotes a* Skolem rule *obtained from* r *as follows: every existential variable* $v \in \mathcal{V}_{H\exists}(r)$ *is substituted by the term* $sk^n_v(s)$ *with* sk^n_v *the Skolem function (constant) symbol associated to* v *and* $n = ar(sk^n_v)$ *the size of* s *(zero if* $\mathcal{V}_{H\forall}(r) = \emptyset$*). The* Skolem program *sk(P) of an* $\exists$*-program* P *is defined by* $sk(P) = \{sk(r) | r \in P\}$*.*

Example 3 (Example 1 continued) *The Skolem rule of* r_2 *is the rule:*

$$
\begin{aligned}
sk(r_2) = (phdS(X, sk^1_D(X)), d(sk^1_D(X)) \leftarrow & \\
p(X), not(l(X), gC(X, Y)).)&
\end{aligned}
$$

Hence $sk(P_U) = \{r_0, r_1, sk(r_2)\}$ *and* $\mathcal{L}_{sk(P_U)} = (\{a\}, \{sk^1_D\}, \{p, phdS, d, l, gC\})$.

Skolem rules are still not safe: existential variables remain in the negative bodies. The grounding of such a rule is a partial grounding restricted to the universal variables of the rule, the existential ones remaining not ground. Indeed, a non-ground rule $(p(X) \leftarrow q(X), not\ r(X, Z).)$ could be fired for some constant a if $q(a)$ is true and, for all z, $r(a, z)$ is not true. Suppose two constants a and b. Then $(p(a) \leftarrow q(a), not\ r(a, a).)$ and $(p(a) \leftarrow q(a), not\ r(a, b).)$ are not equivalent to the non-ground rule for $X = a$ because the first instance could be fired if $r(a, b)$ is true (but not $r(a, a)$) and the second could be fired if $r(a, a)$ is true (but not $r(a, b)$). Yet neither $r(a, b)$ nor $r(a, a)$ should be true for the initial rule to be fired. We thus define a partial grounding, only concerning universal variables. For instance, a partial ground instance of the above non-ground rule would be: $(p(a) \leftarrow q(a), not\ r(a, Z).)$.

Definition 4 (Partial Ground Program) *Set* $\mathbf{PG}(r)$ *for a rule* r *of an* $\exists$*-program* P *of language* $\mathcal{L}_P$ *denotes the set*

of all partial ground instances of r over the language $\mathcal{L}_P$ for $\overline{\mathcal{V}_{N\exists}}(r)$. The partial ground program $\mathbf{PG}(P)$ of an $\exists$-program P is defined by $\mathbf{PG}(P) = \bigcup_{r\in P} \mathbf{PG}(r)$.

Example 4 (Example 1 continued) *The language of the Skolem program $sk(P_U)$ contains only one constant, a, and only one function symbol, sk^1_D. The set of ground terms is infinite and the partial grounding leads then to the following infinite program:*

$$\begin{array}{l}\mathbf{PG}(sk(P_U)) = \{\\ \quad p(a).,\\ \quad l(a).,\\ \quad phdS(a, sk^1_D(a)), d(sk^1_D(a)) \leftarrow\\ \quad\quad p(a), not\ (l(a), gC(a, Y)).,\\ \quad phdS(sk^1_D(a), sk^1_D(sk^1_D(a))), d(sk^1_D(sk^1_D(a))) \leftarrow\\ \quad\quad p(sk^1_D(a)), not\ (l(sk^1_D(a)), gC(sk^1_D(a), Y)).,\\ \quad \ldots\}\end{array}$$

Proposition 2 *The partial ground program of an $\exists$-program with no multiple head, no multiple default negation and no existential variable is a ground (classical) ASP program.*

Proof 2 *This is a direct consequence of Definitions 1 and 4.*

Definition 5 (Reduct) *Let P be an $\exists$-program of language $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. The reduct of the partial ground program $\mathbf{PG}(sk(P))$ w.r.t. X is the definite partial ground program*

$$\begin{array}{l}\mathbf{PG}(sk(P))^X =\\ \{\quad head(r) \leftarrow body^+(r).|r \in \mathbf{PG}(sk(P)),\\ \quad\ \text{for all } N \in body^-(r) \text{ and}\\ \quad\ \text{for all ground substitution } \sigma \text{ over } \mathcal{L}_{sk(P)}, \sigma(N) \notin X\}\end{array}$$

Example 5 (Example 1 continued) *Let*

$$X_1 = \{p(a), l(a), phdS(a, sk^1_D(a)), d(sk^1_D(a))\}.$$

Then, for the rule

$$phdS(a, sk^1_D(a)), d(sk^1_D(a)) \leftarrow p(a), not\ (l(a), gC(a, Y)).$$

there is no ground instance of $l(a)$,$gC(a, Y)$ that is included in X_1 (since X_1 does not contain any atom with gC) and the positive part of the rule is kept. Other rules are kept for the same reason. The obtained program is then:

$$\begin{array}{l}\mathbf{PG}(sk(P_U))^{X_1} = \{\\ \quad p(a).,\\ \quad l(a).,\\ \quad phdS(a, sk^1_D(a)), d(sk^1_D(a)) \leftarrow p(a).,\\ \quad phdS(sk^1_D(a), sk^1_D(sk^1_D(a))), d(sk^1_D(sk^1_D(a))) \leftarrow\\ \quad\quad p(sk^1_D(a)).,\\ \quad \ldots\}\end{array}$$

Now, let $X_2 = X_1 \cup \{gC(a, m)\}$ and $P_{U2} = P_U \cup \{gC(a, m).\}$.

Here, $l(a)$,$gC(a, m)$ is a ground instance of the negative body of the rule

$$phdS(a, sk^1_D(a)), d(sk^1_D(a)) \leftarrow p(a), not\ (l(a), gC(a, Y)).$$

that is included in X_2. Thus, the rule is excluded from the reduct. Other rules are kept. The obtained program is then:

$$\begin{array}{l}\mathbf{PG}(sk(P_U \cup \{gC(a, m).\}))^{X\cup\{gC(a,m)\}} = \{\\ \quad gC(a, m).,\\ \quad p(a).,\\ \quad l(a).,\\ \quad phdS(sk^1_D(a), sk^1_D(sk^1_D(a))), d(sk^1_D(sk^1_D(a))) \leftarrow\\ \quad\quad p(sk^1_D(a)).,\\ \quad \ldots\}\end{array}$$

Note that the reduct of a program that is Skolemized and partially grounded is a definite ground program: it no longer contains variables. The consequence operator can then be defined as usual, the only difference is that rules can have a conjunction of atoms as head.

Definition 6 (T_P consequence operator and Cn its closure) *Let P be a definite partial ground program of an $\exists$-program of language $\mathcal{L}_P$. The operator $T_P : 2^{\mathbf{GA}(\mathcal{L}_P)} \rightarrow 2^{\mathbf{GA}(\mathcal{L}_P)}$ is defined by*

$$T_P(X) = \{a|r \in P, a \in head(r), body^+(r) \subseteq X\}.$$

$Cn(P) = \bigcup_{n=0}^{n=+\infty} T_P^n(\emptyset)$ is the least fix-point of the consequence operator T_P.

Example 6 (Example 1 continued) $Cn(\mathbf{PG}(sk(P_U))^X) = X$ *but* $Cn(\mathbf{PG}(sk(P_U \cup \{gC(a, m).\}))^{X\cup\{gC(a,m)\}}) = \{p(a), l(a), gC(a, m)\}$.

Definition 7 ($\exists$-answer set) *Let P be an $\exists$-program of language $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. X is an $\exists$-answer set of P if and only if $X = Cn(\mathbf{PG}(sk(P))^X)$.*

Example 7 (Example 1 continued) *X is an $\exists$-answer set of P_U and $\{p(a), l(a), gC(a, m)\}$ is an $\exists$-answer set of $P_U \cup \{gC(a, m).\}$.*

Proposition 3 *Let P be a (classical) ASP program of language $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_P)$. X is an answer set of P if and only if X is an $\exists$-answer set of P considered as an $\exists$-program.*

Proof 3 *Since P is a classical ASP program, $sk(P) = P$ and its (classical) ground ASP program corresponds exactly to $\mathbf{PG}(P) = \mathbf{PG}(sk(P))$. Hence $X \in \mathbf{GA}(\mathcal{L}_P) = \mathbf{GA}(\mathcal{L}_{sk(P)})$ is an answer set of ground P, by Definition 7, if and only if it is an $\exists$-answer set of P considered as an $\exists$-program.*

From $\exists$-ASP to ASP

In this section, we give the translation of an $\exists$-ASP program into a standard ASP program and we show that the $\exists$-answer sets of the initial program correspond to the answer sets of the new program.

The first step of the translation is the normalization whose goal is twofold: to remove the conjunctions of atoms from negative parts of the rules and to remove existential variables from these negative parts. The obtained program is equivalent in terms of answer sets.

Definition 8 *Let P be an $\exists$-program of language $\mathcal{L}_P$. Let r be an $\exists$-rule of P ($m, s \geq 0$, $n, u_1, \ldots, u_s \geq 1$):*

$$H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m, not\ (N_1^1, \ldots, N_{u_1}^1), \ldots, not\ (N_1^s, \ldots, N_{u_s}^s).$$

with $H_1, \ldots, H_n, B_1, \ldots, B_m, N_1^1, \ldots, N_{u_1}^1, \ldots, N_1^s, \ldots, N_{u_s}^s \in \mathbf{A}(\mathcal{L}_P)$. Let $\mathcal{N}$ be a set of new predicate symbols (i.e. $\mathcal{N} \cap \mathcal{PS} = \emptyset$).

The normalization *of such an $\exists$-rule is the set of $\exists$-rules*

$$\begin{aligned}&\mathbf{N}(r) = \\ &\{\quad H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m, not\ N_1, \ldots, not\ N_s., \\ &\quad N_1 \leftarrow N_1^1, \ldots, N_{u_1}^1., \\ &\quad \ldots \\ &\quad N_s \leftarrow N_1^s, \ldots, N_{u_s}^s.\}\end{aligned}$$

with N_i the new atom $p^{N_i}(X_1, \ldots, X_v)$, $p^{N_i} \in \mathcal{N}$ a new predicate symbol for every N_i and $\mathcal{V}_\forall(r)(N_1^i, \ldots, N_{u_i}^i) = \{X_1, \ldots, X_v\}$.

The normalization of P is defined as $\mathbf{N}(P) = \bigcup_{r \in P} \mathbf{N}(r)$.

Set $\mathbf{GAN}(\mathcal{L}_{sk(P)})$ is the set of Skolem ground atoms for the new predicate symbols defined as follows:

- *if $a \in \mathcal{N}$ with $ar(a) = 0$ then $a \in \mathbf{GAN}(\mathcal{L}_{sk(P)})$,*
- *if $p \in \mathcal{N}$ with $ar(p) > 0$ and $t_1, \ldots, t_n \in \mathbf{GT}(\mathcal{L}_{sk(P)})$ then $p(t_1, \ldots, t_n) \in \mathbf{GAN}(\mathcal{L}_{sk(P)})$.*

Example 8 (Example 1 continued) *Let p^N be a new predicate symbol. The negative part of the rule r_2: $not(l(X), gC(X, Y))$ has only one universal variable, X. It is replaced by $not\ p^N(X)$ (rule $r_2^\dagger$). And a new rule $r_2^\ddagger$ is added where Y that was an existential variable in r_2 becomes a universal one in $r_2^\ddagger$.*

$$\begin{aligned}&\mathbf{N}(r_2) = \{ \\ &\quad r_2^\dagger : phdS(X, D), d(D) \leftarrow p(X), not\ p^N(X). \\ &\quad r_2^\ddagger : p^N(X) \leftarrow l(X), gC(X, Y).\}\end{aligned}$$

and $\mathbf{N}(P_U) = \{r_0, r_1, r_2^\dagger, r_2^\ddagger\}$.

The following proposition shows that normalization preserves answer sets of an $\exists$-program: it only adds some atoms formed with the new predicate symbols from $\mathcal{N}$.

Proposition 4 *Let P be an $\exists$-program of language $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. If X is an $\exists$-answer set of P then there exists $Y \subseteq \mathbf{GAN}(\mathcal{L}_{sk(P)})$ such that $X \cup Y$ is an $\exists$-answer set of $\mathbf{N}(P)$. If X is an $\exists$-answer set of $\mathbf{N}(P)$ then $X \setminus \mathbf{GAN}(\mathcal{L}_{sk(P)})$ is an $\exists$-answer set of P.*

The lemma used in the following proof shows that if the normalization is applied on only one rule r and only one part of the negative body of this rule, then the answer sets of the original program are preserved up to the added atom. If r has the following form:

$$H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m, not\ (N_1^1, \ldots, N_{u_1}^1), \ldots, not\ (N_1^s, \ldots, N_{u_s}^s).$$

then the "partial normalization" of r for $(N_1^s, \ldots, N_{u_s}^s)$ leads to the rules

$$\begin{aligned}&r^\dagger = H_1, \ldots, H_n \leftarrow \\ &\quad B_1, \ldots, B_m, \\ &\quad not\ (N_1^1, \ldots, N_{u_1}^1), \ldots, not\ (N_1^{s-1}, \ldots, N_{u_{s-1}}^{s-1}), not\ N_s.\end{aligned}$$

and $r^\ddagger = N_s \leftarrow N_1^s, \ldots, N_{u_s}^s$. A program P with the rule r and the program P where the rule r is replaced by the rules $r^\dagger$ and $r^\ddagger$ have the same answer sets except for N_s. The proof can be constructed by induction by applying the lemma to each part of the negative body of r and, then, to each rule of the program.

Proof 4 *The proof is by induction on the following lemma: Let P be an $\exists$-program of language $\mathcal{L}_P$, $r = (H \leftarrow C, not\ (N_1, \ldots, N_u).) \in \mathbf{PG}(sk(P))$, $P' = \mathbf{PG}(sk(P)) \setminus \{r\}$, $r^\dagger = (H \leftarrow C, not\ N.) \in \mathbf{PG}(sk(\mathbf{N}(P)))$, $R^\ddagger = \mathbf{PG}(N \leftarrow N_1, \ldots, N_u.) \subseteq \mathbf{PG}(sk(\mathbf{N}(P)))$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$.*

If there exists a substitution θ such that $\{\theta(N_1), \ldots, \theta(N_u)\} \subseteq X$ then $Cn((P' \cup \{r\})^X) = X$ if and only if $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^{X \cup \{N\}}) = X \cup \{N\}$. If for all substitutions θ, $\{\theta(N_1), \ldots, \theta(N_u)\} \not\subseteq X$ then $Cn((P' \cup \{r\})^X) = X$ if and only if $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^X) = X$.

Let us remark that $N \notin Cn(P'^X) \cup X$.

- *If there exists a substitution θ such that $\{\theta(N_1), \ldots, \theta(N_u)\} \subseteq X$ then $(P' \cup \{r\})^X = P'^X = (P' \cup \{r^\dagger\})^{X \cup \{N\}}$ then $Cn((P' \cup \{r\})^X) = Cn(P'^X)$ and $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^{X \cup \{N\}}) = Cn(P'^X) \cup \{N\}$. Then $Cn((P' \cup \{r\})^X) = X$ iff $Cn(P'^X) = X$ iff $Cn(P'^X) \cup \{N\} = X \cup \{N\}$ iff $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^{X \cup \{N\}}) = X \cup \{N\}$.*
- *If for all substitutions θ, $\{\theta(N_1), \ldots, \theta(N_u)\} \not\subseteq X$ then $(P' \cup \{r\})^X = (P' \cup \{H \leftarrow C.\})^X$ and $(P' \cup \{r^\dagger\} \cup R^\ddagger)^X = (P' \cup \{H \leftarrow C.\})^X \cup R^\ddagger$. Then $Cn((P' \cup \{r\})^X) = Cn((P' \cup \{H \leftarrow C.\})^X) = Cn((P' \cup \{H \leftarrow C.\})^X \cup R^\ddagger) = Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^X)$. Then $Cn((P' \cup \{r\})^X) = X$ iff $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^X) = X$.*

Example 9 (Example 1 continued) *Program P_U, after normalization, is Skolemized and grounded. After normalization and Skolemization, the program no longer contains existential variables. Thus, after grounding, it does not contain any more variables.*

$$\begin{aligned}&\mathbf{PG}(sk(\mathbf{N}(P_U))) = \{ \\ &\quad p(a)., \\ &\quad l(a), \\ &\quad phdS(a, sk_D^1(a)), d(sk_D^1(a)) \leftarrow p(a), not\ p^N(a). \\ &\quad p^N(a) \leftarrow l(a), gC(a, a)., \\ &\quad p^N(a) \leftarrow l(a), gC(a, sk_D^1(a))., \\ &\quad \ldots, \\ &\quad phdS(sk_D^1(a), sk_D^1(sk_D^1(a))), d(sk_D^1(sk_D^1(a))) \leftarrow \\ &\qquad p(sk_D^1(a)), not\ p^N(sk_D^1(a))., \\ &\quad p^N(sk_D^1(a)) \leftarrow l(sk_D^1(a)), gC(sk_D^1(a), a)., \\ &\quad p^N(sk_D^1(a)) \leftarrow l(sk_D^1(a)), gC(sk_D^1(a), sk_D^1(a))., \\ &\quad \ldots\}\end{aligned}$$

The following proposition shows that Skolemization and grounding preserve answer sets of a normalized $\exists$-program.

Proposition 5 *Let P be a normalized $\exists$-program of language $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. X is an $\exists$-answer set of P if and only if X is an $\exists$-answer set of $\mathbf{PG}(sk(P))$.*

Proof 5 *Since for all* $r \in \mathbf{PG}(sk(P)), \mathcal{V}_{N\exists}(r) = \emptyset$ *(since* r *is normalized),* $\overline{\mathcal{V}_{N\exists}}(r) = \mathcal{V}(r)$ *and* $\mathcal{V}_{H\exists}(r) = \emptyset$ *(since* r *is Skolemized) then* $\mathbf{PG}(sk(P))$ = $sk(\mathbf{PG}(sk(P)))$ = $\mathbf{PG}(sk(\mathbf{PG}(sk(P))))$.

By Definition 7, X *is an* $\exists$*-answer set of* P *iff* $X = Cn(\mathbf{PG}(sk(P))^X)$ *iff* $X = Cn(\mathbf{PG}(sk(\mathbf{PG}(sk(P))))^X)$ *iff* X *is an* $\exists$*-answer set of* $\mathbf{PG}(sk(P))$.

Once an $\exists$-program is normalized and Skolemized, the only non-standard parts that remain are the conjunctions of atoms in rule heads. The last step of the translation is the expansion where we remove the sets of atoms in each head while keeping the link between the existential variables. It simply consists to duplicate a rule as many time as the rule contains atoms in its head, each new rule having only one of these atoms in its head. Preceding Skolemization allows to preserve the links between the existential variables of the head. The obtained program is equivalent in terms of answer sets.

Definition 9 *Let* P *be a ground Skolemized normalized program and* $r \in P$ *(*$m, s \geq 0$*,* $n > 0$*):*

$$H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m, not\ N_1, \ldots, not\ N_s.$$

with $H_1, \ldots, H_n, B_1, \ldots, B_m, N_1, \ldots, N_s \in \mathbf{GA}(\mathcal{L}_P)$.

The expansion *of such a rule is the set defined by:*

$$\begin{array}{l}\mathbf{Exp}(r) = \\ \{\quad H_1 \leftarrow B_1, \ldots, B_m, not\ N_1, \ldots, not\ N_s., \\ \quad\quad \ldots \\ \quad\quad H_n \leftarrow B_1, \ldots, B_m, not\ N_1, \ldots, not\ N_s.\}\end{array}$$

The expansion of P *is defined as* $\mathbf{Exp}(P) = \bigcup_{r\in P} \mathbf{Exp}(r)$.

Example 10 (Example 1 continued) *The following rule of the program from Example 9:*

$phdS(a, sk^1_D(a)), d(sk^1_D(a)) \leftarrow p(a), not\ p^N(a).$ *is splitted into the two rules:*

$phdS(a, sk^1_D(a)) \leftarrow p(a), not\ p^N(a).$ *and*

$d(sk^1_D(a)) \leftarrow p(a), not\ p^N(a).$

The same treatment is applied to the other rules with both predicates $phdS$ *and* d *in the head.*

The following program is obtained:

$$\begin{array}{l}\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P_U)))) = \{ \\ \quad p(a)., \\ \quad l(a)., \\ \quad phdS(a, sk^1_D(a)) \leftarrow p(a), not\ p^N(a)., \\ \quad d(sk^1_D(a)) \leftarrow p(a), not\ p^N(a)., \\ \quad p^N(a) \leftarrow l(a), gC(a, a)., \\ \quad p^N(a) \leftarrow l(a), gC(a, sk^1_D(a))., \\ \quad \ldots, \\ \quad phdS(sk^1_D(a), sk^1_D(sk^1_D(a))) \leftarrow \\ \qquad\qquad p(sk^1_D(a)), not\ p^N(sk^1_D(a))., \\ \quad d(sk^1_D(sk^1_D(a))) \leftarrow p(sk^1_D(a)), not\ p^N(sk^1_D(a))., \\ \quad p^N(sk^1_D(a)) \leftarrow l(sk^1_D(a)), gC(sk^1_D(a), a)., \\ \quad p^N(sk^1_D(a)) \leftarrow l(sk^1_D(a)), gC(sk^1_D(a), sk(a))., \\ \quad \ldots\}\end{array}$$

Proposition 6 *Let* P *be a ground Skolemized normalized* $\exists$*-program of language* $\mathcal{L}_P$ *and* $X \subseteq \mathbf{GA}(\mathcal{L}_P)$. X *is an* $\exists$*-answer set of* P *if and only if* X *is an* $\exists$*-answer set of* $\mathbf{Exp}(P)$.

Proof 6 *The only difference is on the computation of the fix-point of (classical)* T_P *operator and new* T_P *operator defined in Definition 6 and clearly enough fix-point are identical since* P *is ground.*

Proposition 7 *Let* P *be an* $\exists$*-program.* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$ *is an (ground classical) ASP program.*

Proof 7 *This proposition is a direct consequence of Definitions 3, 4, 8, 9 and Proposition 2.*

The last proposition establishes equivalence, up to new atoms introduced by normalization, between $\exists$-answer sets of an $\exists$-program and classical answer sets of the program after normalization, Skolemization and expansion.

Proposition 8 *Let* P *be an* $\exists$*-program of language* $\mathcal{L}_P$ *and* $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. *If* X *is an* $\exists$*-answer set of* P *then there exists* $Y \subseteq \mathbf{GAN}(\mathcal{L}_{sk(P)})$ *such that* $X \cup Y$ *is a (classical) answer set of* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$. *If* X *is a (classical) answer set of* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$*, then* $X \setminus \mathbf{GAN}(\mathcal{L}_{sk(P)})$ *is an* $\exists$*-answer set of* P.

Proof 8 *Let* P *be an* $\exists$*-program and* $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$.

- *if* X *is an* $\exists$*-answer set of* P *then, by proposition 4, there exists* $Y \subseteq \mathbf{GAN}(\mathcal{L}_{sk(P)})$ *such that* $X \cup Y$ *is an* $\exists$*-answer set of* $\mathbf{N}(P)$. *By proposition 5,* $X \cup Y$ *is an* $\exists$*-answer set of* $\mathbf{PG}(sk(\mathbf{N}(P)))$. *By proposition 6,* $X \cup Y$ *is an* $\exists$*-answer set of* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$. *By propositions 3 and 7,* $X \cup Y$ *is an answer set of* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$.
- *If* X *is a (classical) answer set of* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$ *then, by propositions 3 and 7,* X *is an* $\exists$*-answer set of* $\mathbf{Exp}(\mathbf{PG}(sk(\mathbf{N}(P))))$. *By proposition 6,* X *is an* $\exists$*-answer set of* $\mathbf{PG}(sk(\mathbf{N}(P)))$. *By proposition 5,* X *is an* $\exists$*-answer set of* $\mathbf{N}(P)$. *By proposition 4,* $X \setminus \mathbf{GAN}(\mathcal{L}_{sk(P)})$ *is an* $\exists$*-answer set of* P.

Conclusion

This paper is a first step of formalisation of ASP allowing the use of existential variables. It is well suited to integrate ontologies and rules in a unique formalism.

From a practical point of view, the proposed translation from $\exists$-ASP to ASP allows us to use any solver. But let us note that we have implemented this translation as a front-end of the solver `ASPeRiX` which uses on-the-fly grounding (Lefèvre et al. 2015). This should help, in the future, for dealing with variables in a more efficient way.

An in-depth comparison with other formalisms remains to be done. One of the closest work is (Baget et al. 2014b) dealing with existential rules extended with non monotonic negation. In this work, existential variables are only allowed in the rule heads, not in the negative bodies. `ASPeRiX` semantics (defined via a notion of computation inspired from (Liu et al. 2010)) is adapted for defining different chases

(forward chaining algorithms) for non monotonic existential rules. Our present work should be linked to one of these chases, the Skolem-chase.

Another ongoing work is to deal efficiently with queries in this framework. This is not obvious due to the nonmonotonic aspect of ASP and the potential inconsistency of an ASP program. It seems that very little work has been done on these aspects.

Acknowledgements

This work received support from ANR (French National Research Agency), ASPIQ project reference ANR-12-BS02-0003.

References

Alviano, M.; Faber, W.; Leone, N.; and Manna, M. 2012. Disjunctive datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory Pract. Log. Program.* 12(4-5):701–718.

Baget, J.-F.; Garreau, F.; Mugnier, M.-L.; and Rocher, S. 2014a. Extending acyclicity notions for existential rules. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic*, 39–44.

Baget, J.-F.; Garreau, F.; Mugnier, M.-L.; and Rocher, S. 2014b. Revisiting Chase Termination for Existential Rules and their Extension to Nonmonotonic Negation. In Konieczny, S., and Tompits, H., eds., *NMR'2014: 15th International Workshop on Non-Monotonic Reasoning*, volume INFSYS Research Report Series.

Cali, A.; Gottlob, G.; Lukasiewicz, T.; Marnette, B.; and Pieris, A. 2010. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *25th Annual IEEE Symposium on Logic in Computer Science (LICS), 2010*, 228–242.

de Bruijn, J.; Pearce, D.; Polleres, A.; and Valverde, A. 2010. A semantical framework for hybrid knowledge bases. *Knowl. Inf. Syst.* 25(1):81–104.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2005. Dlv-hex: Dealing with semantic web under answer-set programming. In *4th International Semantic Web Conference (ISWC 2005) - Posters Track, Galway, Ireland, November 2005. System poster.*

Eiter, T.; Ianni, G.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2008. Combining answer set programming with description logics for the semantic web. *Artif. Intell.* 172(12-13):1495–1539.

Eiter, T.; Fink, M.; Krennwallner, T.; and Redl, C. 2013. hex-programs with existential quantification. In Hanus, M., and Rocha, R., eds., *Declarative Programming and Knowledge Management, KDPD 2013*, volume 8439 of *Lecture Notes in Computer Science*, 99–117.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence* 175:236–263.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K., eds., *Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP'88)*, 1070–1080. Cambridge, Massachusetts: The MIT Press.

Ianni, G.; Eiter, T.; Tompits, H.; and Schindlauer, R. 2005. Nlp-dl: A kr system for coupling nonmonotonic logic programs with description logics. In *The Forth International Semantic Web Conference (ISWC2005)*.

Lee, J., and Palla, R. 2011. Integrating rules and ontologies in the first-order stable model semantics (preliminary report). In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, 248–253.

Lefèvre, C.; Béatrix, C.; Stéphan, I.; and Garcia, L. 2015. Asperix, a first order forward chaining approach for answer set computing. *CoRR* abs/1503.07717:(to appear in TPLP).

Liu, L.; Pontelli, E.; Son, T. C.; and Truszczynski, M. 2010. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence* 174(3-4):295–315.

Magka, D.; Krötzsch, M.; and Horrocks, I. 2013. Computing stable models for nonmonotonic existential rules. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*.

Motik, B., and Rosati, R. 2010. Reconciling description logics and rules. *J. ACM* 57(5).

Rosati, R. 2006. Dl+log: Tight integration of description logics and disjunctive datalog. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, 68–78.

You, J.-H.; Zhang, H.; and Zhang, Y. 2013. Disjunctive logic programs with existential quantification in rule heads. *Theory and Practice of Logic Programming* 13:563–578.