Bringing existential variables in answer set programming and bringing non-monotony in existential rules: two sides of the same coin

Jean-François Baget, Laurent Garcia, Fabien Garreau, Claire Lefèvre, Swan Rocher, Igor Stéphan

Dear Author

Here are the proofs of your article.

- You can submit your corrections **online,** via **e-mail** or by **fax**.
- For **online** submission please insert your corrections in the online correction form. Always indicate the line number to which the correction refers.
- You can also insert your corrections in the proof PDF and **email** the annotated PDF.
- For **fax** submission, please ensure that your corrections are clearly legible. Use a fine black pen and write the correction in the margin, not too close to the edge of the page.
- Remember to note the **journal title**, **article number**, and **your name** when sending your response via e-mail or fax.
- **Check** the metadata sheet to make sure that the header information, especially author names and the corresponding affiliations are correctly shown.
- **Check** the questions that may have arisen during copy editing and insert your answers/corrections.
- **Check** that the text is complete and that all figures, tables and their legends are included. Also check the accuracy of special characters, equations, and electronic supplementary material if applicable. If necessary refer to the *Edited manuscript*.
- The publication of inaccurate data such as dosages and units can have serious consequences. Please take particular care that all such details are correct.
- Please **do not** make changes that involve only matters of style. We have generally introduced forms that follow the journal's style.
- Substantial changes in content, e.g., new results, corrected values, title and authorship are not allowed without the approval of the responsible editor. In such a case, please contact the Editorial Office and return his/her consent together with the proof.
- If we do not receive your corrections **within 48 hours**, we will send you a reminder.
- Your article will be published **Online First** approximately one week after receipt of your corrected proofs. This is the **official first publication** citable with the DOI. **Further changes are, therefore, not possible.**
- The **printed version** will follow in a forthcoming issue.

Please note

After online publication, subscribers (personal/institutional) to this journal will have access to the complete article via the DOI using the URL:

`http://dx.doi.org/10.1007/s10472-017-9563-9`

If you would like to know when your article has been published online, take advantage of our free alert service. For registration and further information, go to: http://www.link.springer.com.

Due to the electronic nature of the procedure, the manuscript and the original figures will only be returned to you on special request. When you return your corrections, please inform us, if you would like to have these documents returned.

AUTH Metadata of the article that will be visualized in OnlineFirst

Please note: Images will appear in color online but will be printed in black and white.

1	Article Title	**Bringing existential variables in answer set programming and bringing non-monotony in existential rules: two sides of the same coin**	
2	Article Sub-Title		
3	Article Copyright - Year	**Springer International Publishing AG 2017 (This will be the copyright line in the final PDF)**	
4	Journal Name	Annals of Mathematics and Artificial Intelligence	
5	Corresponding Author	Family Name	**Garcia**
6		Particle	
7		Given Name	**Laurent**
8		Suffix	
9		Organization	University of Angers
10		Division	LERIA
11		Address	Angers, France
12		e-mail	laurent.garcia@univ-angers.fr
13	Author	Family Name	**Baget**
14		Particle	
15		Given Name	**Jean-François**
16		Suffix	
17		Organization	INRIA
18		Division	
19		Address	Rennes Cedex, France
20		e-mail	jean-francois.baget@lirmm.fr
21	Author	Family Name	**Garreau**
22		Particle	
23		Given Name	**Fabien**
24		Suffix	
25		Organization	University of Angers
26		Division	LERIA
27		Address	Angers, France
28		e-mail	fabien.garreau@univ-angers.fr
29	Author	Family Name	**Lefèvre**
30		Particle	
31		Given Name	**Claire**
32		Suffix	
33		Organization	University of Angers
34		Division	LERIA
35		Address	Angers, France

36		e-mail	claire.lefevre@univ-angers.fr
37	Author	Family Name	**Rocher**
38		Particle	
39		Given Name	**Swan**
40		Suffix	
41		Organization	University of Montpellier
42		Division	LIRMM
43		Address	Montpellier, France
44		e-mail	swan.rocher@lirmm.fr
45	Author	Family Name	**Stéphan**
46		Particle	
47		Given Name	**Igor**
48		Suffix	
49		Organization	University of Angers
50		Division	LERIA
51		Address	Angers, France
52		e-mail	igor.stephan@univ-angers.fr

53	Schedule	Received	
54		Revised	
55		Accepted	

56	Abstract	This article deals with the combination of ontologies and rules by means of existential rules and answer set programming. Existential rules have been proposed for representing ontological knowledge, specifically in the context of Ontology- Based Data Access. Furthermore Answer Set Programming (ASP) is an appropriate formalism to represent various problems issued from Artificial Intelligence and arising when available information is incomplete. The combination of the two formalisms requires to extend existential rules with nonmonotonic negation and to extend ASP with existential variables. In this article, we present the syntax and semantics of Existential Non Monotonic Rules (ENM-rules) using skolemization which join together the two frameworks. We formalize its links with standard ASP. Moreover, since entailment with existential rules is undecidable, we present conditions that ensure the termination of a breadth-first forward chaining algorithm known as the chase and we discuss extension of these results in the nonmonotonic case.

57	Keywords separated by ' - '	Answer set programming - Existential rules - Ontologies - Decidability
58	Foot note information	

JrnlID 10472_ArtID 9563_Proof#1 - 05/08/2017

Ann Math Artif Intell
DOI 10.1007/s10472-017-9563-9

Bringing existential variables in answer set programming and bringing non-monotony in existential rules: two sides of the same coin

Jean-François Baget[1] · Laurent Garcia[2] · Fabien Garreau[2] · Claire Lefèvre[2] · Swan Rocher[3] · Igor Stéphan[2]

Abstract This article deals with the combination of ontologies and rules by means of existential rules and answer set programming. Existential rules have been proposed for representing ontological knowledge, specifically in the context of Ontology- Based Data Access. Furthermore Answer Set Programming (ASP) is an appropriate formalism to represent various problems issued from Artificial Intelligence and arising when available information is incomplete. The combination of the two formalisms requires to extend existential rules with nonmonotonic negation and to extend ASP with existential variables. In this article, we present the syntax and semantics of Existential Non Monotonic Rules (ENM-rules) using skolemization which join together the two frameworks. We formalize its links with standard ASP. Moreover, since entailment with existential rules is undecidable, we present conditions that ensure the termination of a breadth-first forward chaining algorithm known as the chase and we discuss extension of these results in the nonmonotonic case.

✉ Laurent Garcia
laurent.garcia@univ-angers.fr

Jean-François Baget
jean-francois.baget@lirmm.fr

Fabien Garreau
fabien.garreau@univ-angers.fr

Claire Lefèvre
claire.lefevre@univ-angers.fr

Swan Rocher
swan.rocher@lirmm.fr

Igor Stéphan
igor.stephan@univ-angers.fr

Q1 [1] INRIA, Rennes Cedex, France

[2] LERIA, University of Angers, Angers, France

[3] LIRMM, University of Montpellier, Montpellier, France

Keywords Answer set programming · Existential rules · Ontologies · Decidability

Q2 **Mathematics Subject Classification (2010)**

1 Introduction

When dealing with information issued from the web, it is interesting to have a system able to represent ontologies and to reason under them. For many years, several works have been proposed to deal with either of these two aspects but it is now important to join these features in one formalism. The work presented here deals with existential nonmonotonic rules.[1] It presents the two sides of a work. On one hand, it enriches the ASP framework by taking into account existential variables. On the other hand, it consists in introducing nonmonotony in existential rules. The proposed work aims at describing knowledge in a single framework which can lead to useful implementation. The interest of focusing on ASP is that it is a powerful framework for knowledge representation and reasoning, and provides efficient solvers. Moreover, existential rules are suitable to deal with ontological knowledge.

Existential rules (also called Datalog+/-) have been proposed for representing ontological knowledge, specifically in the context of Ontology-Based Data Access, that aims to exploit ontological knowledge when accessing data [10, 14]. These rules allow to assert the existence of unknown individuals, a feature recognized as crucial for representing knowledge in an open domain perspective. Existential rules generalize lightweight description logics, such as DL-Lite and EL [3, 17] and overcome some of their limitations by allowing any predicate arity as well as cyclic structures. Alternatively, those existential variables can be seen as functional terms obtained by skolemization. Existential rules are thus a subset of rules with function symbols for which specific decidability results have been obtained (for instance [8] for saturation-based mechanisms).

Answer Set Programming (ASP) is a very convenient paradigm to represent knowledge in Artificial Intelligence (AI), especially when information is incomplete [11]. It has its roots in nonmonotonic reasoning and logic programming and has led to a lot of works since the seminal paper [26]. Beyond its ability to formalize various problems from AI, ASP provides also an interesting way to practically solve such problems since some efficient solvers are available.

This work presents a way for the treatment of ontologies in Answer Set Programming (ASP). 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 the inclusion with exceptions of concepts in the TBox). The problem for ASP is the presence of existential variables in ontologies.

Then the present work has two sides. On the one side, it proposes a definition of ASP with existential variables. The treatment of these variables is done in terms of skolemization. On the other side, it can be seen as the extension of existential rules with nonmonotonic negation under stable model semantics. Note that the restriction of function symbols to those that encode existential variables allow to benefit from all termination properties obtained for the saturation using existential rules.

[1] The work of this paper is a revised and extended version of the papers [9] and [25].

If we consider the intended semantics of $\exists X\ p(X)$ in ASP, there are two main approaches: (1) one can enumerate all possible values for X, that is $\exists X\ p(X)$ is interpreted as $p(a_1) \vee p(a_2) \vee \ldots$ for all a_i belonging to the considered universe, or (2) one can only say that there is some anonymous individual x_0 such that $p(x_0)$ holds: this corresponds to skolemization. In the first approach, the considered universe is the Herbrand universe, eventually extended with other individuals in the case of open domains. In practice this approach generates a lot of answer sets. If we are only interested by the fact that there exists some individual that verifies property p, but not with which one, skolemization is a good solution: it represents exactly the information of existence of some individual. Coupled with the Unique Name Assumption, the skolemization encounters a problem: Skolem terms can not be identified with some other named individual if necessary. For instance, if skolemized, the following program $\{\exists X\ p(X).,\ p(a).,\ \leftarrow p(X), p(Y), X \neq Y.\}$ has no answer set while one can expect $\{p(a)\}$. Nevertheless skolemization enables to verify that there exists exactly one individual satisfying some property p: $\{\leftarrow not\ p(X).,\ \leftarrow p(X), p(Y), X \neq Y.\}$.

Entailment with existential rules is known to be undecidable [12, 18]. Many sufficient conditions for decidability, obtained by syntactic restrictions, have been exhibited in knowledge representation and database theory (see e.g., the overview in [43]). We focus in this paper on conditions that ensure the termination of a breadth-first forward chaining algorithm, known as the chase in the database literature. Given a knowledge base composed of data and existential rules, the chase saturates the data by application of the rules. When it is ensured to terminate, the information deduced by the rules can be added to the data, which can then be queried like a classical database, thus allowing to benefit from any database optimizations technique. Several variants of the chase have been proposed, which differ in the way they deal with redundant information [20, 22, 41]. It follows that they do not behave in the same way with respect to termination. In the following, when we write the chase, we mean one of these variants. Various acyclicity notions have been proposed to ensure the halting of some chase variants. We propose some extensions of these acyclicity notions, while keeping good complexity properties. We discuss the relevance of the chase variants for nonmonotonic existential rules and further extend acyclicity results obtained for existential rules without negation.

The study of the combination of ontologies and rules is not new [19, 21, 24, 34, 40, 42, 45]. In most of these models, the knowledge base is viewed as an 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, for instance in description logic, and $\mathcal{P}$ describes the rules in terms of a logic program.

The integration of the two formalisms can be separated into three classes [21, 34].

In the first class (like in [21]), 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 made using these special predicates.

The second case (like in [42, 45]) corresponds to an hybrid formalism which integrates DLs and rules in a coherent semantic framework. Predicates of $\mathcal{T}$ can be used in the rules of the program. In [45], the representation of information is separated in two parts, a DL knowledge base and a $Datalog^{\neg\vee}$ program, but there are no rules combining both existential variables and negations: existential variables occur in the DL knowledge base and the negations occur in the program. But default negations are not allowed in the DL part and existential variables are not allowed in the program. Moreover, there are some additional restrictions: for instance, predicates of $\mathcal{T}$ can not be used in the negative body of a rule. A variant of this model, based on guarded rules, is proposed in [31].

The last case integrates DLs and rules in a unique formalism. For instance, de Bruijn et al. [19] 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 description logic information but are not allowed in the head of the rules.

Next to these models, Ferraris et al. [24] 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 [34].

In ASP, the closed domain assumption presumes that all relevant domain elements are present in the program. Open ASP (OASP for short) [31] extends the Herbrand universe with a (finite or infinite) set of new constants. But OASP does not deal explicitly with existential variables: $\exists X\ p(X)$ can be represented by $\{existsp \leftarrow p(X).,\quad \leftarrow not\ existsp.\ p(X) \vee not\ \ p(X).\}$; this program instantiated with individuals of an open domain, can "generate" all answer sets of the form $\{p(a)\}$ where a belongs to the open universe. Then [31] is concerned by restricting the syntax to regain decidability. They define extended forest logic programs (EFOLPs) where one part of the program can use open domain but is stratified, and the other part is only instantiated with the constants of the program.

Nonmonotonic extensions to existential rules were recently considered in [15] with stratified negation, [28] with well-founded semantics and [40] with stable model semantics. In this latter work, the knowledge base is a single one allowing existential variables and default negation in a same rule. It deals with skolemized existential rules and focuses on cases where a finite unique model exists. 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 base then belongs to a particular category of stratified programs. The work is both theoretical and practical but it is concerned with a limited extension of ASP.

Some very recent works deals with kinds of non-monotonic rules with existential variables by translating the initial base into tractable bases (for instance, Alviano et al. [2] uses a second-order translation and [1] uses *Datalog* with non-monotonic atoms) but they do not really focus on a computational solution that can be used in practice. As far as we know, the only works leading to an implementation are those of [32], based on [21], and of [40] which has been applied to information about biochemistry. The systems Shy [37] and Nyaya [28] support skolemized existential variables but not default negation. In [47], some query answering is done on skolemized existential R-acyclic rules using ASP solver `Clasp`.

Section 2 gives the background about First Order Logic (FOL), existential rules and ASP useful for the paper. Then, in Section 3, we define existential nonmonotonic rules, an ASP variant allowing existential variables or, equivalently, a nonmonotonic extension of existential rules and answer sets on this kind of programs are defined. Section 4 gives the links between existential nonmonotonic rules and standard ASP with a method to translate a program expressed with existential nonmonotonic rules into a program expressed in (standard) ASP. Proofs about the transformation are also provided. In Section 5, some properties of different chases are discussed. In Section 6, we propose a tool that allows to extend existing acyclicity conditions ensuring chase termination, while keeping good complexity properties. In Section 7, we discuss the relevance of the chase variants for existential nonmonotonic rules and further extend acyclicity results obtained in the case of rules without default negation.

2 Background

2.1 First order logic background

2.1.1 *Syntax*

A *vocabulary* $\mathcal{L}$ is a triplet $(\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ where $\mathcal{CS}$, $\mathcal{FS}$ and $\mathcal{PS}$ are pairwise disjoint sets, respectively of *constant symbols*, *function symbols* and *predicate names* (or *predicate symbols*). We also consider an infinite countable set $\mathcal{V}$ of *variables*, disjoint with the previous ones. A function *ar* from $\mathcal{PS}$ to $\mathbb{N}$ and from $\mathcal{FS}$ to $\mathbb{N}^*$ associates to each predicate name and function symbol its arity.

Let $\mathcal{X}$ be a set. A *functional term* built from $\mathcal{X}$ is defined inductively as either an element of $\mathcal{X}$, or an object of the form $f(x_1, \ldots, x_k)$ where $f \in \mathcal{FS}$ is a function symbol of arity k and the x_i are functional terms built from $\mathcal{X}$.

The set of *terms* $\mathbf{T}(\mathcal{L})$ denotes the set of all functional terms built from the set $\mathcal{CS} \cup \mathcal{V}$ of constants and variables. The set of *ground terms* $\mathbf{GT}(\mathcal{L})$ denotes the set of all functional terms built from the set $\mathcal{CS}$ of constants.

The set $\mathbf{A}(\mathcal{L})$ denotes the set of *atoms* of a vocabulary, which are of form $p(t_1, \ldots, t_k)$ where $p \in \mathcal{PS}$ is a predicate name of arity k and $t_i \in \mathbf{T}(\mathcal{L})$. An atom is said to be *ground* when all its terms are ground, and it is said to be *function-free* when none of its terms contains a function symbol.

An *atomset* on $\mathcal{L}$ is a (possibly infinite) set of atoms on $\mathcal{L}$. It is said to be *ground* when all its atoms are ground, and *function-free* when all its atoms are function-free.

2.1.2 *Semantics*

An *interpretation* of a vocabulary $\mathcal{L}$ is a pair $I = (\Delta_I, .^I)$ where Δ_I is the *interpretation domain*, $\Delta_I \neq \emptyset$, and the *interpretation function* $.^I$ maps:

- each constant symbol $c \in \mathcal{CS}$ to an element of the domain $c^I \in \Delta_I$;
- each function symbol $f \in \mathcal{FS}$ of arity k to a function $f^I : \Delta_I^k \to \Delta_I$;
- each predicate name $p \in \mathcal{PS}$ of arity k to a subset p^I of Δ_I^k.

Let $\mathcal{A}$ be an atomset and σ be a mapping from $vars(\mathcal{A})$ (the variables appearing in $\mathcal{A}$) to Δ_I. For every term t appearing in $\mathcal{A}$, we define inductively t_σ^I by:

- if $t \in \mathcal{V}$ is a variable, then $t_\sigma^I = \sigma(t)$;
- if $t \in \mathcal{CS}$ is a constant, then $t_\sigma^I = t^I$;
- otherwise, $t = f(t_1, \ldots, t_k)$ where $t \in \mathcal{FS}$ is a function symbol of arity k, and $t_\sigma^I = f^I((t_1)_\sigma^I, \ldots, (t_k)_\sigma^I)$.

We say that an interpretation $(\Delta_I, .^I)$ is a *model* of an atomset $\mathcal{A}$ and note $(\Delta_I, .^I) \vdash \mathcal{A}$ when there exists a mapping σ from $vars(\mathcal{A})$ to Δ_I such that, for every atom $p(t_1, ..., t_k) \in \mathcal{A}$, $((t_1)_\sigma^I, \ldots, (t_k)_\sigma^I) \in p^I$. Such a mapping is called a *proof* that $(\Delta_I, .^I)$ is a model of $\mathcal{A}$. Note that an atomset $\mathcal{A}$ has exactly the same models as the First Order Logic (FOL) formula obtained from the existential closure of the formula $\phi(\mathcal{A})$, where $\phi(\mathcal{A})$ is the conjunction of atoms in $\mathcal{A}$.

An atomset is *satisfiable* when it admits a model (*unsatisfiable* otherwise), *valid* when all its interpretations are models (*invalid* otherwise), and we say that $\mathcal{A}_1$ *entails* $\mathcal{A}_2$ (or that $\mathcal{A}_2$ is a *semantic consequence* of $\mathcal{A}_1$) and note $\mathcal{A}_1 \models \mathcal{A}_2$ when all models of $\mathcal{A}_1$ are also models of $\mathcal{A}_2$.

Finally, let us point out that any atomset is satisfiable (it admits an isomorphic model), and that the only valid atomset is the empty one $\emptyset$.

2.1.3 Substitutions

Let $\mathcal{X} \subseteq \mathcal{V}$ be a set of variables, and $\mathcal{T}$ be a set of terms. A *substitution function* s is a mapping from $\mathcal{X}$ to $\mathcal{T}$. If t is a term, we define inductively as follows the *substitution*, denoted $\sigma(t)$, as the extension of the substitution function to the terms:

- if $t \in \mathcal{X}$, then $\sigma(t) = s(t)$;
- if $t \in \mathcal{V} \setminus \mathcal{X}$ is a variable that is not in $\mathcal{X}$, then $\sigma(t) = t$;
- if $t \in \mathcal{CS}$ is a constant, then $\sigma(t) = t$;
- otherwise, $t = f(t_1, \ldots, t_k)$ where $t \in \mathcal{FS}$ is a function symbol of arity k, and $\sigma(t) = f(\sigma(t_1), \ldots, \sigma(t_k))$.

By extension, if $a = p(t_1, \ldots, t_k)$ is an atom, we note $\sigma(a) = p(\sigma(t_1), \ldots, \sigma(t_k))$, and if $\mathcal{A} = \{a_1, \ldots, a_p\}$ is an atomset, we note $\sigma(\mathcal{A}) = \{\sigma(a_1), \ldots, \sigma(a_p)\}$.

We say that a substitution σ is *ground* when it maps $\mathcal{X}$ to ground terms of $\mathbf{GT}(\mathcal{L})$. 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 $\mathcal{V}$ over a vocabulary $\mathcal{L}$ is a mapping from $\mathcal{V}$ 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 $\mathcal{V}$, $\sigma(t)$ (resp. $\sigma(a)$) is a *partial ground instance* of t (resp. a) w.r.t. the set of variables $\mathcal{V}$.

2.1.4 Homomorphisms

Definition 1 (**Homomorphism**) Let F and Q be two atomsets. An *homomorphism* from $\mathcal{F}$ to $\mathcal{Q}$ is a substitution σ from the variables of Q to the terms of F such that $\sigma(Q) \subseteq F$.

Theorem 1 *Let F be an atomset, and Q be a finite atomset. Then $F \models Q$ iff there exists an homomorphism from Q to F.*

HOMOMORPHISM
Data: Two finite atomsets F and Q.
Result: TRUE if there is an homomorphism from Q to F, FALSE otherwise.
The problem is NP-complete in combined complexity. It becomes polynomial when Q has no variable, or when it has a tree-like structure. The problem is in AC^0 in data complexity.

2.2 Existential rules

2.2.1 Syntax

An *existential rule* is a pair of finite sets of atoms noted $H \leftarrow B$ where H is called the *head* of the rule and B is called its *body*. We call *body variables* of the rules the variables that appear in B, *frontier variables* of the rule the variables that appear both in B and H, and *existential variables* of the rule those appearing only in H. These rules have been studied in the litterature under different names: conceptual graphs rules [46] or Datalog+/- [14]. They have the same form as tuple generating dependencies studied in database theory.

2.2.2 Semantics

We say that an interpretation $(\Delta^I, .^I)$ is a model of an existential rule $H \leftarrow B$ when every proof that $(\Delta^I, .^I)$ is a model of B can be extended to a proof that $(\Delta^I, .^I)$ is a model of $B \cup H$. Note that the existential rule $H \leftarrow B$ has exactly the same models as the FOL formula $\forall \mathbf{x}(\phi(B) \rightarrow (\exists \mathbf{y} \phi(H)))$ where $\mathbf{x}$ are the body variables of the rule, $\mathbf{y}$ its existential variables, and ϕ maps a set of atoms to their conjunction.

2.2.3 Derivations

Let F be an atomset and $H \leftarrow B$ be an existential rule. We say that $H \leftarrow B$ is *applicable* to F if there exists an homomorphism σ from B to F. In that case, the application of $H \leftarrow B$ on F according to σ produces an atomset $\alpha(F, H \leftarrow B, \sigma) = F \cup \sigma(\mathit{fresh}(H))$ where *fresh* is a bijective substitution from the existential variables of H to a set of fresh variables (i.e., new freshly generated variables that appear nowhere else).

Let $\mathcal{R}$ be a set of existential rules and F be an atomset. An $\mathcal{R}$-*derivation* from F is a (possibly infinite) sequence $F = F_0, F_1, \ldots, F_k, \ldots$ of atomsets such that, for $i \geq 1$, there exists some rule $H \leftarrow B \in \mathcal{R}$ and an homomorphism σ from B to F_{i-1} such that $F_i = \alpha(F_{i-1}, H \leftarrow B, \sigma)$. We say that this derivation is from F to F' when $F' = \cup_{i=0}^{\infty} F_i$.

Theorem 2 *Let F and Q be two finite atomsets, and $\mathcal{R}$ be a finite set of existential rules. Then $F, \mathcal{R} \models Q$ iff there exists a finite $\mathcal{R}$-derivation from F to F' such that $F' \models Q$.*

DEDUCTION
Data: Two finite atomsets F and Q, a finite set of existential rules $\mathcal{R}$.
Result: TRUE if $F, \mathcal{R} \models Q$, FALSE otherwise.
The problem is semi-decidable in the general case. For decidable subclasses of function-free existential rules, see for instance [4]. We discuss a particular family of decidable classes in Section 6.

2.3 Answer set programming

In this section, we give the main background of the ASP framework.

2.3.1 Program

In ASP, a problem is described in term of a logic program with default negation.

A *normal logic program* (or simply *program*) is a set of *rules* like

$$(c \leftarrow a_1, \ \ldots, \ a_n, \ not \ b_1, \ \ldots, \ not \ b_m.) \quad n \geq 0, m \geq 0 \tag{1}$$

where $c, a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms.

For a rule r (or by extension for a set of rules), we define:

- $head(r) = c$ its *head*,
- $body^+(r) = \{a_1, \ldots, a_n\}$ its *positive body*
- $body^-(r) = \{b_1, \ldots, b_m\}$ its *negative body* and
- $\mathcal{V}(r)$ the set of its variables.

The intuitive meaning of such a rule is: "if all the a_i's are true and it may be assumed that all the b_j's are false then one can conclude that c is true". Symbol *not* denotes the *default* *negation*. A rule with no default negation is a *definite rule* otherwise it is a *nonmonotonic rule*. A program with only definite rules is a *definite logic program*. A program is a *propositional program* if all the predicate symbols are of arity 0. The rules of the program must be *safe*; that is all variables that appear in a rule also appear in the positive part of its body. All the variables are considered to be universally quantified. In the sequel, universally quantified variables will be called *universal variables*.

For each program P, we consider that the set $\mathcal{CS}$ (resp. $\mathcal{FS}$ and $\mathcal{PS}$) consists of all constant (resp. function and predicate) symbols appearing in P.

Let r be a rule and θ a ground substitution over the vocabulary of the program, a rule $\theta(r)$ is a *ground instance* of r. The program P (with variables) can be seen as an intensional version of the program $ground(P)$ defined as follows: given a rule r, $ground(r)$ is the set of all ground instances of r and then, $ground(P) = \bigcup_{r \in P} ground(r)$. Program $ground(P)$ may be considered as a propositional program.

Example 1 The program

$$P_{1a} = \left\{ \begin{array}{l} n(1).,\ n(2).,\\ a(X) \leftarrow n(X),\ not\ b(X).,\\ b(X) \leftarrow n(X),\ not\ a(X). \end{array} \right\}$$

can be seen as a shorthand for the program

$$ground(P_{1a}) = \left\{ \begin{array}{l} n(1).,\ n(2).,\\ a(1) \leftarrow n(1),\ not\ b(1).,\\ b(1) \leftarrow n(1),\ not\ a(1).,\\ a(2) \leftarrow n(2),\ not\ b(2).,\\ b(2) \leftarrow n(2),\ not\ a(2). \end{array} \right\}$$

The program

$$P_{1b} = \left\{ \begin{array}{l} p(a).,\\ l(a).,\\ phdS(X, f(X)) \leftarrow p(X), not(l(X), gC(X, Y)). \end{array} \right\}$$

can be seen as a shorthand for the (infinite) program

$$ground(P_{1b}) = \left\{ \begin{array}{l} p(a).,\\ l(a).,\\ phdS(a, f(a)) \leftarrow p(a), not(l(a), gC(a, a)).\\ phdS(f(a), f(f(a))) \leftarrow p(f(a)), not(l(f(a)), gC(f(a), a)).\\ \dots \end{array} \right\}$$

The following program says that every man X has a father $f(X)$ who is himself a man.

$$P_{1c} = \left\{ \begin{array}{l} man(a).,\\ father(X, f(X)) \leftarrow man(X).,\\ man(f(X)) \leftarrow man(X). \end{array} \right\}$$

JrnlID 10472_ArtID 9563_Proof#1 - 05/08/2017

It can be seen as a shorthand for the (infinite) program

$$ground(P_{1c}) = \left\{\begin{array}{l} man(a).,\\ father(a, f(a)) \leftarrow man(a).,\\ man(f(a)) \leftarrow man(a).\\ father(f(a), f(f(a))) \leftarrow man(f(a)).,\\ man(f(f(a))) \leftarrow man(f(a)).\\ \ldots \end{array}\right\}$$

The immediate consequence operator for a definite logic program P is $T_P : 2^{\mathcal{A}} \to 2^{\mathcal{A}}$ such that $T_P(X) = \{\sigma(head(r)) \mid r \in P,\ \exists\sigma$ a ground substitution s.t. $\sigma(body^+(r)) \subseteq X\}$. The *least Herbrand model* of P is the smallest set of atoms closed under P (denoted $Cn(P)$), i.e., the smallest set X such that $T_P(X) \subseteq X$. It can be computed as the least fixed point of the consequence operator T_P.

2.3.2 *Answer set*

The solutions of the problem encoded by a program are the answers of the program and are called answer sets.

The *reduct* P^X of a normal logic program P w.r.t. an atomset $X \subseteq \mathcal{A}$ is the definite logic program defined by:

$$P^X = \{\sigma(head(r)) \leftarrow \sigma(body^+(r)). \mid r \in P,\ \exists\sigma \text{ a ground substitution over } \mathcal{V}(r) \text{ s.t.}\\ \sigma(body^-(r)) \cap X = \emptyset\}$$

and it is the core of the definition of an *answer set*.

Definition 2 (**Answer Set**) [26] Let P be a normal logic program and X an atomset. X is an answer set of P if $X = Cn(P^X)$.

For instance, the propositional program $\{a \leftarrow not\ b.,\ \ b \leftarrow not\ a.\}$ has two answer sets $\{a\}$ and $\{b\}$.

Example 2 Taking again the program P_{1a}, $ground(P_{1a})$ has four answer sets:

$$\{a(1), a(2), n(1), n(2)\},\ \ \{a(1), b(2), n(1), n(2)\},\\ \{a(2), b(1), n(1), n(2)\},\ \ \{b(1), b(2), n(1), n(2)\}$$

that are thus the answer sets of P_{1a}.

There is another definition of an answer set for a normal logic program based on the notion of *generating rules* which are the rules participating to the construction of the answer set. These rules are important in our approach because they are exactly the rules fired in the ASPeRiX computation presented in the next section.

Definition 3 (**Generating Rules**) Let P be a normal logic program and X be an atomset. $GR_P(X)$, the set of *generating rules* of P, is defined as $GR_P(X) = \{\sigma(r) \mid r \in P, \sigma$ is a ground substitution over $\mathcal{V}(r)$ s.t. $\sigma(body^+(r)) \subseteq X$ and $\sigma(body^-(r)) \cap X = \emptyset\}$.

Definition 4 (**Founded**) A set of ground rules R is *founded* if there exists an enumeration $\langle r_1, \ldots, r_i, \ldots\rangle$ of the rules of R such that $\forall i \geq 1$, $body^+(r_i) \subseteq head\{r_j \mid j < i\}$.

Theorem 3 [36] *Let P be a normal logic program and X be an atomset. Then, X is an answer set of P if and only if $X = head(GR_P(X))$ and $GR_P(X)$ is founded.*

2.3.3 Special rules

In addition to standard rules, ASP can handle special rules to represent constraints and classical negation. Special headless rules, called *constraints*, are admitted and considered equivalent to rules like $(bug \leftarrow \ldots, \ not \ bug.)$ where bug is a new symbol appearing nowhere else. For instance, the program $\{a \leftarrow not\ b.,\ \ b \leftarrow not\ a.,\ \leftarrow a.\}$ has one, and only one, answer set $\{b\}$ because constraint $(\leftarrow a.)$ prevents a to be in an answer set.

When dealing with default negation, we call a *literal* an atom, a, or the negation of an atom, $not\ a$. A literal a is said to be *positive*, and $not\ a$ is said to be *negative*. The corresponding atom a of a literal l is denoted by $at(l)$. For a literal l where $at(l) = a$, let us denote $pred(l)$ the function such that $pred(not\ a) = pred(a) = p$ with p the predicate symbol of the atom a.

For purposes of knowledge representation, one may have to use conjointly strong negation (like $\neg a$) and default negation (like $not\ a$) inside a same program. This is possible in ASP by means of an *extended logic program* [27] in which rules are built with *classical* literals (i.e. an atom a or its strong negation $\neg a$) instead of atoms only. Semantics of extended logic programs distinguishes inconsistent answer sets from absence of answer set. But, if we are not interested in inconsistent answer sets, the semantics associated to an extended logic program is reducible to answer set semantics for a normal logic program using constraints by taking into account the following conventions:

- every classical literal $\neg x$ is encoded by the atom nx,
- for every atom x, the constraint $(\leftarrow x,\ nx.)$ is added.

By this way, only consistent answer sets are kept. In this article, we do not focus on strong negation and literal will never stand for classical literal.

Let us note that one can also use some particular atoms for (in)equalities and simple arithmetic calculus on (positive and negative) integers. Arithmetic operations are treated as a functional arithmetic and comparison relations are treated as built-in predicates.

2.3.4 Computation

In this section, a constructive characterization of answer sets for first-order normal logic programs, based on a concept of `ASPeRiX` *computation* [35, 36], is presented. This concept is itself based on an abstract notion of *computation* for ground programs proposed in [39]. This computation fundamentally uses a forward chaining of rules. It builds incrementally the answer set of the program and does not require the whole set of ground atoms from the beginning of the process. So, it is well suited to deal directly with first order rules by instantiating them during the computation.

The only syntactic restriction required by this methodology is that every rule of a program must be *safe*. That is, all variables occurring in the head or in the negative body of a rule must occur also in its positive body. Note that this condition is already required by all standard evaluation procedures. Moreover, every constraint (i.e. headless rule) is considered given with the particular head $\perp$ and is also safe.

An `ASPeRiX` *computation* is defined as a process on a computation state based on a *partial interpretation* which is defined as follows.

Definition 5 (**Partial Interpretation**) A *partial interpretation* for a program P is a pair $\langle IN, OUT\rangle$ of disjoint atomsets included in the Herbrand base of P.

Intuitively, all atoms in IN belong to a search answer set and all atoms in OUT do not. The notion of partial interpretation defines different status for rules.

Definition 6 (**Rule Status**) Let r be a rule, σ be a ground substitution over $\mathcal{V}(r)$ and $I = \langle IN, OUT\rangle$ be a partial interpretation.

- $\sigma(r)$ is *supported* w.r.t. I when $body^+(\sigma(r)) \subseteq IN$,
- $\sigma(r)$ is *blocked* w.r.t. I when $body^-(\sigma(r)) \cap IN \neq \emptyset$,
- $\sigma(r)$ is *unblocked* w.r.t. I when $body^-(\sigma(r)) \subseteq OUT$,
- r is *applicable* with σ w.r.t. I when $\sigma(r)$ is supported and not blocked.[2]

An ASPeRiX computation is a forward chaining process that instantiates and fires one unique rule at each iteration according to two kinds of inference: a monotonic step of *propagation* and a nonmonotonic step of *choice*. Firing a rule means adding the head of the rule to the set IN.

Definition 7 (Δ_{pro} **and** Δ_{cho}) Let P be a set of first order rules, I be a partial interpretation and R be a set of ground rules.

- $\Delta_{pro}(P, I, R) = \{(r, \sigma) \mid r \in P, \sigma$ is a ground substitution over $\mathcal{V}(r)$ s.t. $\sigma(r)$ is supported and unblocked, and $\sigma(r) \notin R\}$.
- $\Delta_{cho}(P, I, R) = \{(r, \sigma) \mid r \in P, \sigma$ is a ground substitution over $\mathcal{V}(r)$ s.t. $\sigma(r)$ is applicable and $\sigma(r) \notin R\}$.

It is important to notice that the two sets defined above, like the set $ground(P)$, do not need to be explicitly computed. It is in accordance with the fact that we want to avoid their extensive construction. When necessary, a first-order rule r of P can be selected and grounded with propositional atoms occurring in IN and OUT in order to define a new (not already occurring in R) fully ground rule $\sigma(r)$ member of Δ_{pro} or Δ_{cho}. Because of the safety constraint on rules this full grounding is always possible. The sets Δ_{pro} and Δ_{cho} are used in the following definition of an ASPeRiX computation. The specific case of constraints (rules with $\perp$ as head) is treated by adding $\perp$ into OUT set. By this way, if a constraint is fired (violated), $\perp$ should be added into IN and thus, $\langle IN, OUT\rangle$ would not be a partial interpretation.

Definition 8 (ASPeRiX Computation) Let P be a first order normal logic program. An *ASPeRiX computation* for P is a sequence $\langle R_i, I_i\rangle_{i=0}^{\infty}$ of ground rule sets R_i and partial interpretations $I_i = \langle IN_i, OUT_i\rangle$ that satisfies the following conditions:

- $R_0 = \emptyset$ and $I_0 = \langle\emptyset, \{\perp\}\rangle$,
- (Revision)

  (Propagation) $R_i = R_{i-1} \cup \{r_i\}$ with $r_i = \sigma(r)$ for $(r, \sigma) \in \Delta_{pro}(P, I_{i-1}, R_{i-1})$ and $I_i = \langle IN_{i-1} \cup \{head(r_i)\}, OUT_{i-1}\rangle$

[2]The negation of blocked, *not blocked*, is different from *unblocked*.

or (Rule choice) $\Delta_{pro}(P, I_{i-1}, R_{i-1}) = \emptyset$, $R_i = R_{i-1} \cup \{r_i\}$ with $r_i = \sigma(r)$ for $(r, \sigma) \in \Delta_{cho}(P, I_{i-1}, R_{i-1})$ and $I_i = \langle IN_{i-1} \cup \{head(\sigma_i(r_i))\}, OUT_{i-1} \cup body^-(\sigma_i(r_i))\rangle$

or (Stability) $R_i = R_{i-1}$ and $I_i = I_{i-1}$,

– (Convergence) $IN_\infty = \bigcup_{i=0}^{\infty} IN_i = T'_P(IN_\infty)$[3]

where $T'_P(X) = \{a \mid \exists r \in ground(P), head(r) = a, body^+(r) \subseteq X, body^-(r) \cap X = \emptyset\}$. The computation is said to converge to the set IN_∞.

Example 3 Let P_3 be the following program:

$$\left\{\begin{array}{l} R_1 : n(1). \\ R_2 : n(X+1) \leftarrow n(X), (X+1) <= 2. \\ R_3 : a(X) \leftarrow n(X), not\ b(X), not\ b(X+1). \\ R_4 : b(X) \leftarrow n(X), not\ a(X). \\ R_5 : c(X) \leftarrow n(X), not\ b(X+1). \end{array}\right\}$$

The following sequence is an `ASPeRiX` computation for P_3:

$$\begin{array}{ll}
I_0 & = \langle \emptyset, \{\bot\}\rangle \\
\\
r_1 & = n(1).\ \text{with } (R_1, \emptyset) \in \Delta_{pro}(P_3, I_0, \emptyset) \\
I_1 & = \langle \{n(1)\}, \{\bot\}\rangle \\
\\
r_2 & = n(2) \leftarrow n(1).\ \text{with } (R_2, \{X \leftarrow 1\}) \in \Delta_{pro}(P_3, I_1, \{r_1\}) \\
I_2 & = \langle \{n(1), n(2)\}, \{\bot\}\rangle \\
\\
 & \Delta_{pro}(P_3, I_2, \{r_1, r_2\}) = \emptyset \\
r_3 & = a(1) \leftarrow n(1), not\ b(1), not\ b(2).\ \text{with } (\{R_3, X \leftarrow 1\}) \in \Delta_{cho}(P_3, I_2, \{r_1, r_2\}) \\
I_3 & = \langle \{n(1), n(2), a(1)\}, \{\bot, b(1), b(2)\}\rangle \\
\\
r_4 & = c(1) \leftarrow n(1), not\ b(2).\ \text{with } (\{R_5, X \leftarrow 1\}) \in \Delta_{pro}(P_3, I_3, \{r_1, r_2, r_3\}) \\
I_4 & = \langle \{n(1), n(2), a(1), c(1)\}, \{\bot, b(1), b(2)\}\rangle \\
\\
 & \Delta_{pro}(P_3, I_4, \{r_1, r_2, r_3, r_4\}) = \emptyset \\
r_5 & = a(2) \leftarrow n(2), not\ b(2), not\ b(3).\ \text{with } (\{R_3, X \leftarrow 2\}) \in \Delta_{cho}(P_3, I_4, \{r_1, r_2, r_3, r_4\}) \\
I_5 & = \langle \{n(1), n(2), a(1), c(1), a(2)\}, \{\bot, b(1), b(2), b(3)\}\rangle \\
\\
r_6 & = c(2) \leftarrow n(2), not\ b(3).\ \text{with } (\{R_5, X \leftarrow 2\}) \in \Delta_{pro}(P_3, I_5, \{r_1, r_2, r_3, r_4, r_5\}) \\
I_6 & = \langle \{n(1), n(2), a(1), c(1), a(2), c(2)\}, \{\bot, b(1), b(2), b(3)\}\rangle \\
\\
 & \Delta_{pro}(P_3, I_6, \{r_1, r_2, r_3, r_4, r_5, r_6\}) = \emptyset \\
 & \Delta_{cho}(P_3, I_6, \{r_1, r_2, r_3, r_4, r_5, r_6\}) = \emptyset \\
I_7 & = I_6 \\
 & IN_\infty = \{n(1), n(2), a(1), c(1), a(2), c(2)\} = T'_{P_3}(IN_\infty)
\end{array}$$

[3] In [36], convergence is only guaranteed for finite ground programs and is expressed by: $\exists i \geq 0$, $\Delta_{cho}(P, I_i, R_i) = \emptyset$. The condition $IN_\infty = T'_P(IN_\infty)$ enables to deal with infinite cases.

The previous ASPeRiX computation converges to the set $\{n(1), n(2), a(1), c(1), a(2), c(2)\}$ which is an answer set for P_3.

The following theorem establishes a connection between the results of any ASPeRiX computation and the answer sets of a normal logic program.

Theorem 4 [36] *Let P be a normal logic program and X be an atomset. Then, X is an answer set of P if and only if there is an* ASPeRiX *computation $\langle R_i, I_i \rangle_{i=0}^{\infty}$, $I_i = \langle IN_i, OUT_i \rangle$, for P such that $IN_\infty = X$.*

Let us note that the use of function symbols leads to an infinite Herbrand universe and, besides, leads to an infinite ground program. Without functions symbols, there is an exact correspondence between computations that halts and answer sets. But, when functions symbols are introduced, some computations do not necessarily halt. For instance, a computation can clearly not halt if the computed answer set is infinite. It is the case for the Program P_{1c} from Example 1. On the other hand, Program P_{1b} from Example 1 has an infinite grounding but computations halt without problem.

2.4 Limits of existential rules and ASP

When dealing with ontologies expressed in description logic, the use of ASP can enrich the model by allowing to represent information with exceptions through the default negation. However, ASP does not cover the whole features of description logic. For instance, even in the most restricted version of description logic like DL-Lite, some concepts called *existential concepts* require the use of existential variables. These variables lead to release the safety constraint of the rules. When dealing with such an information, a rule can contain existential variables which do not appear in the positive body of the rule.

On the other hand, existential rules which are suitable to deal with *existential concepts* cannot handle default reasoning since they can be seen as definite rules. The scope of representation is then smaller than the one offered by ASP.

The standard ASP formalism as the existential rules formalism must then be enriched: ASP by allowing non-safe rules to cover existential rules and existential rules by allowing default negation to cover non monotonicity.

3 Syntax and semantics of existential non-monotonic rules

To improve the capacity of representation, we define a new formalism allowing to represent both existential rules and rules of ASP in the same framework. Such new rules are called `existential non-monotonic rules` (ENM-rules or ENMR, for short) since they contain both existential variables in the head of the rule and default negation in its body.

These ENM-rules are of the form:

$$h_1, \ldots, h_n \leftarrow b_1, \ldots, b_m, not\ (n_1^1, \ldots, n_{u_1}^1), \ldots, not\ (n_1^s, \ldots, n_{u_s}^s).$$

where $h_1, \ldots, h_n, b_1, \ldots, b_m, n_1^1, \ldots, n_{u_1}^1, \ldots, n_1^s, \ldots, n_{u_s}^s$ are atoms.

We can note that ENM-rules extend existential rules by allowing the use of default negation in the body.

Moreover, ENM-rules extend classical safe rules of ASP. Let us recall that safety imposes that all variables that appear in a rule also appear in the positive part of its body. In a safe rule, all variables are interpreted as universally quantified. These classical ASP rules are extended in two ways. First, the safety condition is relaxed by allowing atoms from the head and the negative body of a rule to contain variables that do not appear in the positive part of the rule. These variables are interpreted as existential ones. Second, the head of the rule is replaced by a conjunction of atoms and each negated atom is also replaced by a conjunction of atoms. These conjunctions allow multiple atoms to refer to the same existential variable.

For example, in the ENM-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 9 (ENM-**rule and** ENM-**program**) An ENM-program P of vocabulary $\mathcal{L} = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ is a set of ENM-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 $h_1, \ldots, h_n, b_1, \ldots, b_m, n_1^1, \ldots, n_{u_1}^1, \ldots, n_1^s, \ldots, n_{u_s}^s \in \mathbf{A}(\mathcal{L})$.

We use the following notations:

- $head(r) = \{h_1, \ldots, h_n\}$.
- $body^+(r) = \{b_1, \ldots, b_m\}$.
- $body^-(r) = \{\{n_1^1, \ldots, n_{u_1}^1\}, \ldots, \{n_1^s, \ldots, n_{u_s}^s\}\}$.
- $\mathcal{V}(r)$ 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),
- $\mathcal{V}_\exists(r)(n_1^i, \ldots, n_{u_i}^i)$ variables which are in $n_1^i, \ldots, n_{u_i}^i$ but not in $b_1, \ldots, b_m$, $1 \leq i \leq s$ (i.e. existential variables of $n_1^i, \ldots, n_{u_i}^i$).
- $\mathcal{V}_{N\exists}(r) = \bigcup_{1 \leq i \leq s} \mathcal{V}_\exists(r)(n_1^i, \ldots, n_{u_i}^i)$,
- $\overline{\mathcal{V}_{N\exists}}(r) = \mathcal{V}(r) \setminus \mathcal{V}_{N\exists}(r)$,
- $\mathcal{V}_\exists(r) = \mathcal{V}_{H\exists}(r) \bigcup \mathcal{V}_{N\exists}(r)$
- $\mathcal{V}_{H\forall}(r)$ 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).
- $\mathcal{V}_\forall(r)(n_1^i, \ldots, n_{u_i}^i)$ the variables which are at least in $n_1^i, \ldots, n_{u_i}^i$ and in $b_1, \ldots, b_m$ (i.e. universal variables of $n_1^i, \ldots, n_{u_i}^i$).

Moreover, the sets $\mathcal{V}_\exists(r)(n_1^i, \ldots, n_{u_i}^i)$ 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 $n_1^i, \ldots, n_{u_i}^i$ or if it appears in $h_1, \ldots, h_n$ and in one of the $n_1^i, \ldots, n_{u_i}^i$, $1 \leq i \leq s$, 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 4 Let P_U be an ENM-program of vocabulary $\mathcal{L}_U = (\{a\}, \emptyset, \{p, phdS, d, l, gC\})$ 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.

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

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 vocabulary $\mathcal{L}_P = (\mathcal{CS}, \mathcal{FS}, \mathcal{PS})$ consists of exactly the constant symbols, function symbols and predicate symbols appearing in P.

The semantics of ENM-programs uses skolemization of existential variables appearing in the heads of the rules. We now define this skolemization.

Definition 10 (**Skolem symbols**) Let r be an ENM-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_Y^n is a Skolem function symbol of arity n (if $n = 0$ then sk_Y is a Skolem constant symbol).

Example 5 (**Example 4 continued**) Symbol sk_D^1 is a Skolem function symbol of arity 1 for the existential variable D of the head of the rule r_2.

Definition 11 (**Skolem Program**) Let P be an ENM-program of vocabulary $\mathcal{L}_P$.

Let s be an ordered sequence of the variables $\mathcal{V}_{H\forall}(r)$ of an ENM-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 $sk_v^n(s)$ with sk_v^n the Skolem function (constant) symbol associated to v and $n = ar(sk_v^n)$ 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 $sk(P) = \{sk(r) \mid r \in P\}$.

Example 6 (**Example 4 continued**) The Skolem rule of r_2 is the rule:

$sk(r_2) = (phdS(X, sk_D^1(X)), d(sk_D^1(X)) \leftarrow p(X), not(l(X), gC(X, Y)).)$

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

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. Let us suppose that we have only 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 12 (**Partial Ground Program**) Set $\mathbf{PG}(r)$ for a rule r of an ENM-program P of vocabulary $\mathcal{L}_P$ denotes the set of all partial ground instances of r over the vocabulary $\mathcal{L}_P$ for $\overline{\mathcal{V}_{N\exists}}(r)$. The partial ground program $\mathbf{PG}(P)$ of an ENM-program P is defined by $\mathbf{PG}(P) = \bigcup_{r \in P} \mathbf{PG}(r)$.

Example 7 (**Example 4 continued**) The vocabulary 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:

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

Definition 13 (**Reduct**) Let P be an $\exists$-program with vocabulary $\mathcal{L}_P$ 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

$$
\begin{array}{l}
\mathbf{PG}(sk(P))^X = \\
\{\ head(r) \leftarrow body^+(r).|\ r \in \mathbf{PG}(sk(P)),\\
\qquad\qquad\qquad \text{for all } N \in body^-(r) \text{ and}\\
\qquad\qquad\qquad \text{for all ground substitution } \sigma \text{ over } \mathcal{L}_{sk(P)}, \sigma(N) \not\subseteq X\}
\end{array}
$$

Example 8 (**Example 4 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. The other rules are kept for the same reason. The resulting program is then:

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

Now, let $X_2 = X_1 \cup \{gC(a, m)\}$ and the augmented program $P_U \cup \{gC(a, m).\}$. Here, $l(a)$,$gC(a, m)$ is a ground instance of the negative body of the rule

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

JrnlID 10472_ArtID 9563_Proof#1 - 05/08/2017

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

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

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

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

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

Example 9 (**Example 4 continued**) $Cn(\mathbf{PG}(sk(P_U))^{X_1}) = X_1$ but $Cn(\mathbf{PG}(sk(P_U \cup \{gC(a,m).\}))^{X_1 \cup \{gC(a,m)\}}) = \{p(a), l(a), gC(a,m)\}$.

Definition 15 (**∃-answer set**) Let P be an ENM-program of vocabulary $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. X is an ∃-answer set of P if $X = Cn(\mathbf{PG}(sk(P))^X)$.

Example 10 (**Example 4 continued**) X_1 is an ∃-answer set of P_U and $\{p(a), l(a), gC(a,m)\}$ is an ∃-answer set of $P_U \cup \{gC(a,m).\}$.

The two following propositions establish that ENM-programs are extensions of ASP programs and existential rules. They are direct consequences of Definitions 9 and 12.

Proposition 1 *Any (first-order classical) ASP program is an ENM-program. And any set of existential rules is an ENM-program.*

Proposition 2 *The partial ground program of an ENM-program without conjunction of atoms in the head nor on a default negation, and without existential variable is a ground (classical) ASP program; and it is also a set of ground existential rules.*

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

Proof 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 $X \subseteq \mathbf{GA}(\mathcal{L}_P) = \mathbf{GA}(\mathcal{L}_{sk(P)})$ is an answer set of ground P, by Definition 15, if and only if it is an ∃-answer set of P considered as an ENM-program. □

4 Translation to ASP

In this section, we give the translation of an ENM-program into a standard ASP program and we show that the ∃-answer sets of the initial program correspond to the answer sets of the new program. The translation operates in 3 main stages: first, the rules are normalized in order to remove multiple atoms and existential variables from their negative bodies; second, rules are skolemized in order to remove existential variables from their heads; third, rules are expanded in order to remove multiple atoms from their heads.

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 16 (Normalization) Let P be an ENM-program of vocabulary $\mathcal{L}_P$. Let r be an ENM-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 ENM-rule is the set of ENM-rules

$$\begin{aligned}\mathbf{N}(r) = \{\ & h_1, \ldots, h_n \leftarrow b_1, \ldots, b_m, not\ n_1, \ldots, not\ n_s.,\\ & n_1 \leftarrow n_1^1, \ldots, n_{u_1}^1.,\\ & \ldots\\ & n_s \leftarrow n_1^s, \ldots, n_{u_s}^s.\}\end{aligned}$$

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 $\mathbf{N}(P) = \bigcup_{r \in P} \mathbf{N}(r)$.

The 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 11 (**Example 4 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 $r_2^\ddagger$.

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

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

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

JrnlID 10472_ArtID 9563_Proof#1 - 05/08/2017

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

$$r^\dagger = h_1, \ldots, h_n \leftarrow b_1, \ldots, b_m, not\ (n_1^1, \ldots, n_{u_1}^1), \ldots, not\ (n_1^{s-1}, \ldots, n_{u_{s-1}}^{s-1}), not\ n_s.$$
$$r^\ddagger = n_s \leftarrow n_1^s, \ldots, n_{u_s}^s.$$

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 is done by induction: by applying the lemma to each part of the negative body of r and, then, to each rule of the program.

Proof The proof is by induction on the following lemma:
(*) Let P be an ENM-program of vocabulary $\mathcal{L}_P$, $r = (H \leftarrow C, not\ (n_1, \ldots, n_u).) \in \mathbf{PG}(sk(P))$, $P' = \mathbf{PG}(sk(P)) \setminus \{r\}$, $R^\ddagger = \mathbf{PG}(n \leftarrow n_1, \ldots, n_u.) \subseteq \mathbf{PG}(sk(\mathbf{N}(P)))$, $r^\dagger = (H \leftarrow C, not\ n.)$ 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 $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^X) = X$.

Proof of Lemma (*): 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 $Cn((P' \cup \{r^\dagger\} \cup R^\ddagger)^X) = X$.

The proof is completed by successively applying the lemma (*) to each part of the negative body of each rule of the program: it shows that $\exists$-answer sets of P and $\mathbf{N}(P)$ are the same except for the new predicates from $\mathbf{GAN}(\mathcal{L}_{sk(P)})$. $\square$

After normalization, the second step of the translation consists in skolemizing the program. After normalization and skolemization, the program no longer contains existential variables. It can then be grounded and therefore no longer contains any variable.

644 *Example 12* (**Example 4 continued**) Program P_U, after normalization, is skolemized and
645 grounded.

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

646 The following proposition shows that skolemization and grounding preserve answer sets
647 of a normalized ENM-program.

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

650 *Proof* Since for all $r \in \mathbf{PG}(sk(P))$, $\mathcal{V}_{N\exists}(r) = \emptyset$ (since r is normalized), $\overline{\mathcal{V}_{N\exists}}(r) =$
651 $\mathcal{V}(r)$ and $\mathcal{V}_{H\exists}(r) = \emptyset$ (since r is skolemized) then $\mathbf{PG}(sk(P)) = sk(\mathbf{PG}(sk(P))) =$
652 $\mathbf{PG}(sk(\mathbf{PG}(sk(P))))$.
653 By Definition 15, X is an $\exists$-answer set of P iff $X = Cn(\mathbf{PG}(sk(P))^X)$ iff $X =$
654 $Cn(\mathbf{PG}(sk(\mathbf{PG}(sk(P))))^X)$ iff X is an $\exists$-answer set of $\mathbf{PG}(sk(P))$. □

655 Once an ENM-program is normalized and skolemized, the only non-standard parts that
656 remain are the conjunctions of atoms in rule heads. The last step of the translation is the
657 expansion where we remove the sets of atoms in each head while keeping the link between
658 the existential variables. It simply consists in the duplication of a rule as many time as the
659 rule contains atoms in its head, each new rule having only one of these atoms in its head.
660 Preceding skolemization allows to preserve the links between the existential variables of the
661 head. The resulting program is equivalent in terms of answer sets.

662 **Definition 17** (**Expansion**) Let P be a ground skolemized normalized program and $r \in P$
663 ($m, s \geq 0, n > 0$):

$$h_1, \ldots, h_n \leftarrow b_1, \ldots, b_m, not\ n_1, \ldots, not\ n_s.$$

664 with $h_1, \ldots, h_n, b_1, \ldots, b_m, n_1, \ldots, n_s \in \mathbf{GA}(\mathcal{L}_P)$.
665 The *expansion* of such a rule is the set defined by:

$$\begin{array}{rl}
\mathbf{Exp}(r) = \{ & h_1 \leftarrow b_1, \ldots, b_m, not\ n_1, \ldots, not\ n_s.,\\
& \ldots\\
& h_n \leftarrow b_1, \ldots, b_m, not\ n_1, \ldots, not\ n_s.\}
\end{array}$$

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

Example 13 (Example 4 continued) The following rule of the program from Example 12: $(phdS(a, sk_D^1(a)), d(sk_D^1(a)) \leftarrow p(a), not\ p^N(a).)$ is split into the two rules: $(phdS(a, sk_D^1(a)) \leftarrow p(a), not\ p^N(a).)$ and $(d(sk_D^1(a)) \leftarrow p(a), not\ p^N(a).)$
The same treatment is applied to the other rules with both predicates $phdS$ and d in the head. The following program is obtained:

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

Proposition 6 *Let P be a ground skolemized normalized* ENM*-program of vocabulary $\mathcal{L}_P$ 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 The only difference is on the computation of the fixed point of the classical T_P operator and the new T_P operator defined in Definition 14 but it is clear that fixed points are identical since P is ground. □

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

Proof This proposition is a direct consequence of Definitions 11, 12, 16, 17 and Proposition 2. □

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

Proposition 8 *Let P be an* ENM*-program of vocabulary $\mathcal{L}_P$ and $X \subseteq \mathbf{GA}(\mathcal{L}_{sk(P)})$. If X is an $\exists$-answer set of P then there exists some $Y \subseteq \mathbf{GAN}(\mathcal{L}_{sk(P)})$ 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 Let P be an ENM-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.

□

In the next sections, we go back to the existential rules side. We present variants of a breadth-first forward chaining algorithm known as the chase. Since entailment with existential rules is undecidable, we present conditions that ensure the termination of the chase and we discuss extension of these results for the ENM-rules.

5 Discussion of the chase procedures

Let us now consider a derivation from F as defined in Section 2.2.3. Rule applications may add *redundancy*. For instance, if $F = \{p(a)\}$ and $R = \{q(Y) \leftarrow p(X).\}$, we can obtain a derivation $F = F_0, F_1 = \{p(a), q(Y_0)\}, F_2 = \{p(a), q(Y_0), q(Y_1)\}$. Since F_1 and F_2 are semantically equivalent, any atomset that can be obtained by a derivation from F_2 will be equivalent to an atomset that can be obtained by a derivation from F_1.

An algorithm that computes an $\mathcal{R}$-derivation by exploring all possible rule applications in a breadth-first manner is called a *chase*. In the following, we will also call chase the derivation it computes. Different kinds of chase can be defined by using different properties to compute $F_i' = \sigma_i(F_i)$ in the derivation (hereafter we write F_i' for $\sigma_i(F_i)$ when there is no ambiguity). All these algorithms are sound and complete w.r.t. the ENTAILMENT problem in the sense that $(F, \mathcal{R}) \models Q$ iff they provide in finite (but unbounded) time a finite $\mathcal{R}$-derivation from F to F_k such that $F_k \models Q$.

5.1 Different kinds of chase

In the *oblivious chase* (also called naive chase), e.g., [13], a rule R is applied according to an homomorphism π only if it has not already been applied according to the same homomorphism. Let $F_i = \alpha(F_{i-1}', R, \pi)$, then $F_i' = F_{i-1}'$ if R was previously applied according to π, otherwise $F_i' = F_i$. This can be slightly improved. Two applications π and π' of the same rule add the same atoms if they map frontier variables identically (for any frontier variable x of R, $\pi(x) = \pi'(x)$); we say that they are frontier-equal. In the *frontier chase*, let $F_i = \alpha(F_{i-1}', R, \pi)$. We take $F_i' = F_{i-1}'$ if R was previously applied according to some π' frontier-equal to π, otherwise $F_i' = F_i$.

The *Skolem chase* [41] relies on a skolemisation of the rules: a rule R is transformed into a rule $skolem(R)$ by replacing each occurrence of an existential variable Y with a functional term $f_Y^R(\mathbf{X})$, where $\vec{X}$ are the frontier variables of R. Then the oblivious chase is run on skolemized rules. This is the derivation we have considered in this paper. It can easily be checked that frontier chase and Skolem chase yield isomorphic results, in the sense that they generate exactly the same atomsets, up to a bijective renaming of variables by Skolem terms.

The *restricted chase* (also called standard chase) [22] detects a kind of local redundancy. Let $F_i = \alpha(F'_{i-1}, R, \pi)$, then $F'_i = F_i$ if π is useful,[4] otherwise $F'_i = F'_{i-1}$. A slight improvement would be the *piece-restricted chase*. Let $F_i = \alpha(F'_{i-1}, H \leftarrow B., \pi)$. Let P be the maximal subset of H such that $\alpha(F'_{i-1}, P \leftarrow B., \pi)$ is not useful. Then we take $F'_i = \alpha(F'_{i-1}, (H \setminus P) \leftarrow B., \pi)$.

The *core chase* [20] considers the strongest possible form of redundancy: for any F_i, F'_i is the core of F_i.[5]

A chase is said to be *local* if $\forall i \leq j$, $F'_i \subseteq F'_j$. All chase variants presented above are local, *except for the core chase*. This property will be critical for nonmonotonic existential rules.

5.2 Chase termination

Since ENTAILMENT is undecidable, the chase may not halt. We call *C-chase* a chase relying on some criterion C to generate $\sigma(F_i) = F'_i$. So C can be oblivious, skolem, restricted, core or any other criterion that ensures the equivalence between F_i and F'_i. A C-chase generates a possibly infinite $\mathcal{R}$-derivation $\sigma_0(F), \sigma_1(F_1), \ldots, \sigma_k(F_k), \ldots$

We say that this derivation *produces* the (possibly infinite) atomset $(F, \mathcal{R})^C = \cup_{0 \leq i \leq \infty} \sigma_i(F_i) \setminus \cup_{0 \leq i \leq \infty} \overline{(\sigma_i(F_i))}$, where $\overline{(\sigma_i(F_i))} = F_i \setminus \sigma(F_i)$. Note that this produced atomset is usually defined as the infinite union of the $\sigma_i(F_i)$. Both definitions are equivalent when the criterion C is *local*. But the usual definition would produce too big an atomset with a non-local chase such as the core chase: an atom generated at step i and removed at step j would still be present in the infinite union. We say that a (possibly infinite) derivation obtained by the C-chase is *complete* when any further rule application on that derivation would produce the same atomset. A complete derivation obtained by any C-chase produces a *universal model* (i.e., most general) of $(F, \mathcal{R})$: for any atomset Q, we have $F, \mathcal{R} \models Q$ iff $(F, \mathcal{R})^C \models Q$.

We say that the C-chase *halts* on $(F, \mathcal{R})$ when the C-chase generates a finite complete $\mathcal{R}$-derivation from F to F_k. Then $(F, \mathcal{R})^C = \sigma_k(F_k)$ is a finite universal model. We say that $\mathcal{R}$ is *universally C-terminating* when the C-chase halts on $(F, \mathcal{R})$ for any atomset F. If a set of rules is universally C-terminating, we say it is *C-finite*, and we also call C-finite, by extension, the class of C-finite sets of rules. It is well known that the chase variants do not behave in the same way w.r.t. termination. The following examples highlight these different behaviors.

Example 14 (**Oblivious / Skolem chase**) Let $R = p(X, Z) \leftarrow p(X, Y).$ and $F = \{p(a, b)\}$. The oblivious chase does not halt: it adds $p(a, Z_0)$, $p(a, Z_1)$, etc. The frontier chase adds $p(a, Z_0)$ then stops. The skolem chase considers the rule $p(X, f^R_Z(X)) \leftarrow p(X, Y).$; it adds $p(a, f^R_Z(a))$ then halts.

Example 15 (**Skolem / Restricted chase**) Let $R : r(X, Y), r(Y, Y), p(Y) \leftarrow p(X).$ and $F = \{p(a)\}$. The skolem chase does not halt: at Step 1, it maps X to a and adds

[4] Given a rule $R = H \leftarrow B.$, a homomorphism π from B to F is said to be *useful* if it cannot be extended to a homomorphism from $B \cup H$ to F

[5] An atomset F is a *core* if there is no homomorphism from F to one of its strict subsets. Among all atomsets equivalent to an atomset F, there exists a unique core (up to isomorphism). We call this atomset *the* core of F.

$r(a, f_Y^R(a))$, $r(f_Y^R(a), f_Y^R(a))$ and $p(f_Y^R(a))$; at step 2, it maps X to $f_Y^R(a)$ and adds $r(f_Y^R(a), f_Y^R(f_Y^R(a)))$, etc. The restricted chase performs a single rule application, which adds $r(a, Y_0)$, $r(Y_0, Y_0)$ and $p(Y_0)$; indeed, the rule application that maps X to Y_0 yields only redundant atoms w.r.t. $r(Y_0, Y_0)$ and $p(Y_0)$.

Example 16 (**Restricted / Core chase**) Let $F = \{s(a)\}$, $R_1 = p(X, X_1), p(X, X_2), r(X_2, X_2) \leftarrow s(X).$, $R_2 = q(Y) \leftarrow p(X, Y).$ and $R_3 = r(X, Y), q(Y) \leftarrow q(X).$. Note that R_1 creates redundancy and R_3 could be applied indefinitely if it were the only rule. R_1 is the first applied rule, which creates new variables, called X_1 and X_2 for simplicity. The restricted chase does not halt: R_3 is not applied on X_2 because it is already satisfied at this point, but it is applied on X_1, which creates an infinite chain. The core chase applies R_1, computes the core of the result, which removes $p(a, X_1)$, then halts.

It is natural to consider the oblivious chase as the weakest form of chase (without the oblivious criterion, any rule having an existential variable would generate an infinite number of instantiations of that variable), and necessary to consider the core chase as the strongest form of chase (since the core is the minimal representative of its equivalence class). We say that a criterion C is *stronger* than C' and write $C \succeq C'$ when C'-finite $\subseteq C$-finite. We say that C is *strictly stronger* than C' (and write $C \succ C'$) when $C \succeq C'$ and $C' \not\succeq C$.

Consider a breadth-first derivation $D = (F_0, F_1, \dots, F_k, \dots)$ that relies upon the weaker oblivious chase. Then consider two chase criterions X and Y. We can thus consider the derivations $D^X = (F_0^X, F_1^X, \dots, F_k^X, \dots)$ and $D^Y = (F_0^Y, F_1^Y, \dots, F_k^Y, \dots)$ where, $\forall 1 \leq i$, $F_i^X = \sigma_i^X(F_i)$ and $F_i^Y = \sigma_i^Y(F_i)$ are obtained by the simplification mechanisms of X and Y. We say that X is stronger than Y on D if $\forall 1 \leq i$, $F_i^X \subseteq F_i^Y$. We say that X is stronger than Y (and write $X \geq Y$) when, for any such D, X is stronger than Y on D. The following property is immediate.

Property 1 If $X \geq Y$, then Y-finite $\subseteq X$-finite.

We say that X is *strictly stronger* than Y (and note $X > Y$) when $X \geq Y$ and $Y \not\geq X$. We would like to obtain a property of the form "if $X > Y$, then Y-finite is a strict subclass of X-finite". This property does not hold in the general case. Let us consider for instance a *k-lazy-core-chase* that only computes cores every k derivation steps. It is immediate to check that core $\geq$ k-lazy-core. However, core-finite and k-lazy-core-finite are the same class.

The next property expresses that if a chase relies upon a stronger way to simplify atomsets, then it halts on more instances.

Property 2 If X and Y are two local chases such that $X > Y$, then Y-finite $\subset X$-finite.

It is well-known that core > restricted > skolem > oblivious (see for instance [9]). Moreover, the frontier chase and the skolem chase halt on the same instances: π maps the frontier of R in a new way and produces a new atom in the frontier chase iff $\alpha(F, skolem(R), \pi)$ contains a new atom. Thus skolem = frontier.

One can easily check that core > piece-restricted > restricted. It is immediate to check that core $\geq$ piece-restricted $\geq$ restricted. These comparisons are strict since (1) the piece-restricted chase is local and the core chase is not, and (2) the restricted chase does not halt on $(\{p(a, b)\}, \{p(Z, X), r(X, Y) \leftarrow p(X, Y).\}$, but the piece-restricted chase does (it can fold $p(Z, X)$ even if $r(X, Y)$ cannot).

Note that the frontier chase does not fit nicely into this framework: when we consider than X is stronger than Y, we consider the same set of rules $\mathcal{R}$, whereas the frontier-chase considers a skolemization of $\mathcal{R}$. However, we can easily check that the frontier chase and the skolem chase produce isomorphic results: π maps the frontier of R in a new way if and only if $\alpha(F, \mathit{skolem}(R), \pi)$ contains a new atom. Then frontier-finite and skolem-finite are the same class.

An immediate remark is that core-finite corresponds to *finite expansion sets (fes)* defined in [5]. In turn, *fes* correspond to rules enjoying the *bounded derivation depth property* (BDDP) introduced in [14] (see [6] for a proof). To sum up, the following inclusions hold between C-finite classes: oblivious-finite $\subset$ skolem-finite = frontier-finite $\subset$ restricted-finite $\subset$ core-finite = fes.

6 Decidability

Ensuring chase termination has been widely studied, in particular various "acyclicity" notions have been defined ensuring finiteness of the chase. We first give an overview of known acyclicity notions. They can be divided into two main families, each of them relying on a different graph: a "position-based" approach, which intuitively relies on a graph encoding variable sharing between positions in predicates; and a "rule dependency approach" which relies on a graph encoding dependencies between rules, i.e., the fact that a rule may lead to trigger another (or itself).

Position-based approach In the first approach, cycles identified as dangerous are those passing through positions that may contain existential variables; such a cycle meaning that the creation of an existential variable in a given position may lead to create another existential variable in the same position, hence a possibly infinite number of existential variables. In the Skolem chase this may lead to an infinitely deep functional symbol. Acyclicity is then defined by the absence of dangerous cycles. The simplest notion of acyclicity in this family is that of weak-acyclicity (wa) [22, 23] , which has been widely used in databases. It relies on a directed graph whose nodes are the positions in predicates (we denote by (p, i) the position i in predicate p). Then for each rule $R : H \leftarrow B$, and each variable X in B occurring in position (p, i), edges with origin (p, i) are built as follows: if X is a frontier variable, there is an edge from (p, i) to each position of X in H; furthermore for each existential variable Y in H occuring in position (q, j), there is a *special* edge from (p, i) to (q, j). A set of rules if weakly acyclic if its associated graph has no cycle passing through a special edge.

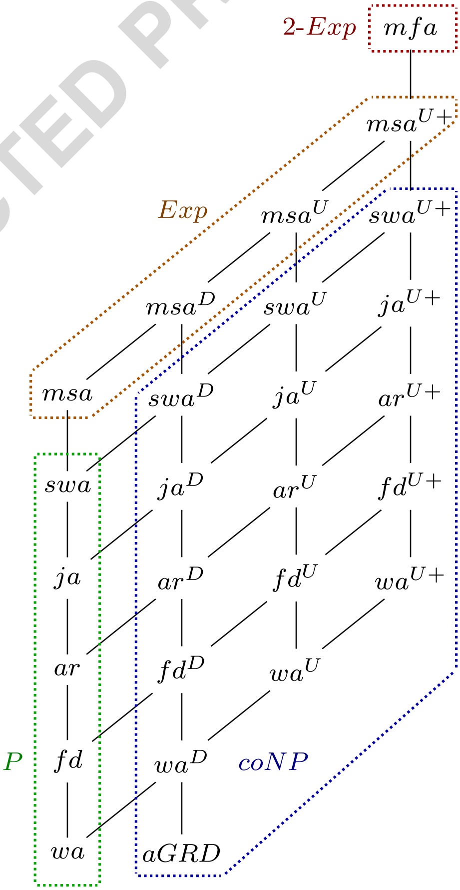
This notion has been generalised, mainly by shifting the focus from positions to existential variables (joint-acyclicity (ja) [33]), or to positions in atoms instead of predicates (super-weak-acyclicity (swa) [41]). Other related notions can be imported from logic programming, e.g., finite domain (fd) [16], and argument-restricted (ar) [38].

Rule dependency approach In the second approach, the aim is to avoid cyclic triggering of rules [7, 20, 29]. We say that a rule R_j *depends* on a rule R_i if there exists an atomset F such that R_i is applicable to F according to a homomorphism π and R_j is applicable to $F' = \alpha(F, R_i, \pi)$ according to a new useful homomorphism. This abstract dependency relation can be computed with a unification operation known as piece-unifier [10]. Piece-unification takes existential variables into account, hence is more complex than the usual unification between atoms. A piece-unifier of a rule body B_j with a rule head H_i is a substitution μ of $vars(B'_j) \cup vars(H'_i)$, where $B'_j \subseteq B_j$, $H'_i \subseteq H_i$, such that: $\mu(B'_j) = \mu(H'_i)$ and existential variables in H'_i are not unified with separating variables of B'_j, i.e., variables that occur both in B'_j and in $B_j \setminus B'_j$; in other words, if a variable X in B'_j is unified with an existential variable Y in H'_i, then all atoms in which X occurs also belong to B'_j. It holds that R_j depends on R_i iff there is a piece-unifier of B_j with H_i satisfying easy to check additional conditions (atom erasing [4], and usefulness [30]).

The *graph of rule dependencies* of set of rules $\mathcal{R}$, denoted by $GRD(\mathcal{R})$, is the directed graph with set of nodes $\mathcal{R}$ and an edge (R_i, R_j) if R_j depends on R_i. When the GRD is acyclic (aGRD [7]), any derivation sequence is necessarily finite. This notion is incomparable with those based on positions (Fig. 1).

Toward a more general point of view Both approaches have their weaknesses: there may be a dangerous cycle on positions but no cycle w.r.t. rule dependencies, and there may be a cycle w.r.t. rule dependencies whereas rules contain no existential variables. Attempts

Fig. 1 Relations between rule classes



to combine both notions only succeeded to combine them in a "modular way": if the rules in each strongly connected component (s.c.c.) of the GRD belong to a class ensuring finiteness of the chase, then the chase will halt on any fact given this set of rules. In the following, we propose an "integrated" way to combining both approaches, which relies on a single graph.

We first define the notion of basic position graph, that encodes precisely how variables in a given position in the body can be propagated to another position of the head by the application of a single rule. Let us consider the graph composed of the basic position graphs for all rules in a given ruleset. We must now now add edges to this graph, encoding how variables added by a given rule may be used by another one (*i.e.*, edges from head positions of rules to body positions of other rules). The graph obtained must be correct: if there exists a variable that propagates in a given derivation, then it corresponds to an edge that must be present in our graph (a precise definition is given below, it considers more correct graphs since it only requires cyclic propagations to be encoded by a cycle in the graph). The goal is now to obtain a correct graph having as few edges as possible (the less edges we consider, the more chances we have to obtain a circuit-free graph and thus to conclude on termination).

We define here three position graphs with increasing expressivity, i.e., allowing to check termination for increasingly larger classes of rules. All these graphs rely upon the notion of position in an atom, and we denote by $[a, i]$ the i^{th} position of atom a.

Definition 18 (**Position Graph** ($\mathcal{PG}$)) The position graph of an ENM-Rule $R : H \leftarrow B$ is the directed graph $\mathcal{PG}(R)$ defined as follows:

- there is a node for each $[a, i]$ in B or in H;
- for all frontier positions $[b, i]$ in B, and all $[h, j]$ in H, there is an edge from $[b, i]$ to $[h, j]$ if $term([b, i]) = term([h, j])$ or if $term([h, j])$ is an existential variable.

In other words, there is an edge from a position in the body to a position in the head when they share a frontier variable, and an edge from each position in the body containing a frontier variable to each position in the head containing an existential variable.

Given a set of ENM rules $\mathcal{R}$, the basic position graph of $\mathcal{R}$ denoted by $\mathcal{PG}(\mathcal{R})$ is the disjoint union of $\mathcal{PG}(R_i)$ for all $R_i \in \mathcal{R}$.

We say that a position $[a, i]$ is *infinite* if $term([a, i])$ is an existential variable, and there is an atomset F such that running the chase on F produces an unbounded number of instantiations of $term([a, i])$. To detect infinite positions, we encode how variables may be propagated between rules by adding edges to $\mathcal{PG}(\mathcal{R})$, called *transition edges*, which go from positions in rule heads to position in rule bodies. The set of transition edges has to be *correct*: if a position $[a, i]$ is infinite, there must be a cycle going through $[a, i]$ in the graph. Though the existence of a transition edge does not necessarily mean that there exists a derivation that will propagate a variable through that edge, its absence in a correct graph means that no possible derivation will ever propagate a variable in such a way.

We then define three position graphs by adding transition edges to $\mathcal{PG}(\mathcal{R})$, namely $\mathcal{PG}^F(\mathcal{R}), \mathcal{PG}^D(\mathcal{R}), \mathcal{PG}^U(\mathcal{R})$. All have correct sets of transition edges. Intuitively $\mathcal{PG}^F(\mathcal{R})$ corresponds to the case where all rules are supposed to depend on all rules; $\mathcal{PG}^D(\mathcal{R})$ encodes actual paths or rule dependencies; and finally, $\mathcal{PG}^U(\mathcal{R})$ adds information about the piece-unifier themselves, providing an accurate encoding of variable propagation from an atom position to another.

Definition 19 ($\mathcal{PG}^X$) Let $\mathcal{R}$ be a set of rules. The three following position graphs are obtained from $\mathcal{PG}(\mathcal{R})$ by adding a (transition) edge from each position $[h, k]$ in a rule head H_i to each position $[b, k]$ in a rule body B_j, with the same predicate, provided that some condition is satisfied:

- full PG, denoted by $\mathcal{PG}^F(\mathcal{R})$: no additional condition;
- dependency PG, denoted by $\mathcal{PG}^D(\mathcal{R})$: if R_j depends directly or indirectly on R_i, i.e., if there is a path from R_i to R_j in $GRD(\mathcal{R})$;
- PG with unifiers, denoted by $\mathcal{PG}^U(\mathcal{R})$: if there is a piece-unifier μ of B_j with the head of an agglomerated rule (see Definition 20) R_i^j such that $\mu(term([b, k])) = \mu(term([h, k]))$.

Example 17 (PG^F **and** PG^D) Let $\mathcal{R} = \{R_1, R_2\}$ with $R_1 = p(X, Y) \leftarrow h(X)$ and $R_2 = h(V) \leftarrow p(U, V), q(V)$. Figure 2 pictures $\mathcal{PG}^F(\mathcal{R})$ and $\mathcal{PG}^D(\mathcal{R})$. The dashed edges belong to $\mathcal{PG}^F(\mathcal{R})$ but not to $\mathcal{PG}^D(\mathcal{R})$. Indeed, R_2 does not depend on R_1. $\mathcal{PG}^F(\mathcal{R})$ has a cycle while $\mathcal{PG}^D(\mathcal{R})$ has not.

Example 18 (PG^D **and** PG^U) Let $\mathcal{R} = \{R_1, R_2\}$, with $R_1 = p(Z, Y), q(Y) \leftarrow t(X, Y)$ and $R_2 = t(V, W) \leftarrow p(U, V), q(U)$. In Fig. 3, the dashed edges belong to $\mathcal{PG}^D(\mathcal{R})$ but not to $\mathcal{PG}^U(\mathcal{R})$. Indeed, the only piece-unifier of B_2 with H_1 unifies U and Y. Hence, the cycle in $\mathcal{PG}^D(\mathcal{R})$ disappears in $\mathcal{PG}^U(\mathcal{R})$.

Definition 20 (**Agglomerated Rule**) Given R_i and R_j rules from $\mathcal{R}$, an agglomerated rule associated with (R_i, R_j) has the following form:

$$R_i^k = H_i \leftarrow B_i \bigcup_{t \in T \subseteq terms(H_i)} fr(t)$$

where fr is a new unary predicate that does not appear in $\mathcal{R}$, and the atoms $fr(t)$ are built as follows. Let $\mathcal{P}$ be a non-empty set of paths from R_i to direct predecessors of R_j in $GRD(\mathcal{R})$. Let $P = (R_1, \ldots, R_n)$ be a path in $\mathcal{P}$. One can associate a rule R^P with P by building a sequence $R_1 = R_1^P, \ldots, R_n^P$ such that $\forall 1 \leq l \leq n$, there is a piece-unifier μ_l of B_{l+1} with the head of R_l^P, where the body of R_{l+1}^p is $B_l^P \cup \{fr(t) \mid t$ is a term of H_l^P unified in $\mu_l\}$, and the head of R_{l+1}^P is H_1. Note that for all l, $H_l^P = H_1$, however, for $l \neq 1$,

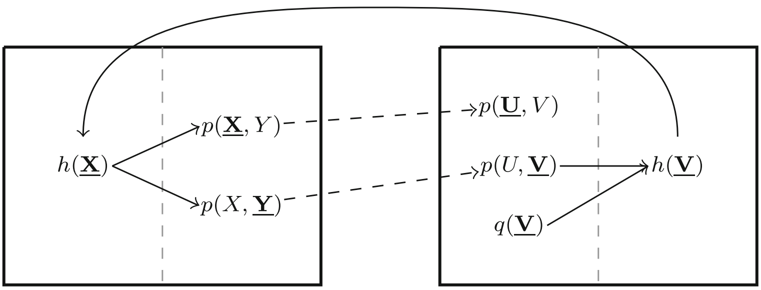


Fig. 2 $\mathcal{PG}^F(\mathcal{R})$ and $\mathcal{PG}^D(\mathcal{R})$ from Example 17. Position $[a, i]$ is represented by underlining the i-th term in a. *Dashed edges* do not belong to $\mathcal{PG}^D(\mathcal{R})$

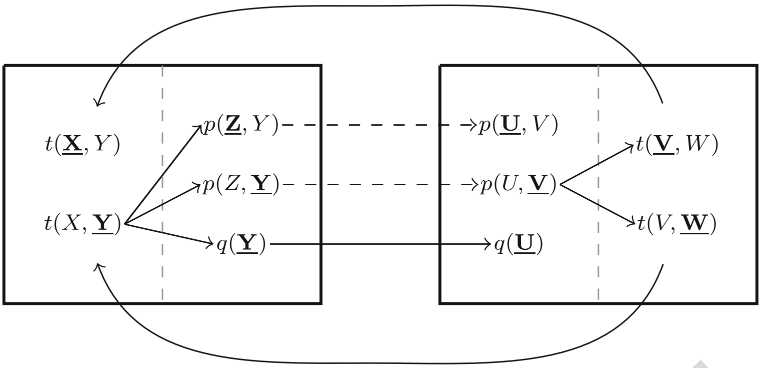


Fig. 3 $\mathcal{PG}^D(\mathcal{R})$ and $\mathcal{PG}^U(\mathcal{R})$ from Example 18. *Dashed edges* do not belong to $\mathcal{PG}^U(\mathcal{R})$

R_l^P may have less existential variables than R_l due to the added atoms. The agglomerated rule R_i^j built from $\{R^P \mid P \in \mathcal{P}\}$ is $R_i^j = \bigcup\limits_{P \in \mathcal{P}} R^P$.

Proposition 9 (**Inclusions between** $\mathcal{PG}^X$) *Let $\mathcal{R}$ be a set of rules. $\mathcal{PG}^U(\mathcal{R}) \subseteq \mathcal{PG}^D(\mathcal{R}) \subseteq \mathcal{PG}^F(\mathcal{R})$. Furthermore, $\mathcal{PG}^D(\mathcal{R}) = \mathcal{PG}^F(\mathcal{R})$ if the transitive closure of $GRD(\mathcal{R})$ is a complete graph.*

We now study how acyclicity properties can be expressed on position graphs. The idea is to associate, with an acyclicity property, a function that assigns to each position a subset of positions reachable from this position, according to some propagation constraints; then, the property is fulfilled if no existential position can be reached from itself. More precisely, a marking function Y assigns to each node $[a, i]$ in a position graph PG^X, a subset of its (direct or indirect) successors, called its marking. A marked cycle for $[a, i]$ (w.r.t. X and Y) is a cycle C in $\mathcal{PG}^X$ such that $[a, i] \in C$ and for all $[a', i'] \in C$, $[a', i']$ belongs to the marking of $[a, i]$. Obviously, the less situations there are in which the marking may "propagate" in a position graph, the stronger the acyclicity property is (in the sense that this property will detect more terminating instances).

Definition 21 (**Acyclicity property**) Let Y be a marking function and $\mathcal{PG}^X(\mathcal{R})$ be a position graph for a set of rules $\mathcal{R}$. The acyclicity property associated with Y in $\mathcal{PG}^X(\mathcal{R})$, denoted by Y^X, is satisfied by $\mathcal{R}$ if there is no marked cycle for any existential position in $\mathcal{PG}^X(\mathcal{R})$. If Y^X is satisfied, we also say that $\mathcal{PG}^X(\mathcal{R})$ satisfies Y.

When there is no ambiguity on the set of rules $\mathcal{R}$ considered, we may note $\mathcal{PG}^X$ instead of $\mathcal{PG}^X(\mathcal{R})$. Note also that in the following, we denote in the same way the property Y^X and the class Y^X of instances that satisfy Y^X (thus conflating the property with the set of instances satisfying the property). It allows us to write, for instance, $Y^X \subseteq Y^Z$ when all instances satisfying Y^X also satisfy Y^Z.

Note that all known rule classes between wa and swa can be expressed as marking functions on the position graph.

The next propositions rely on the following lemma, that makes the link between PG^D and the GRD of a set of rules.

Lemma 1 *Let $\mathcal{R}$ be a set of rules, and Y be an acyclicity property. $\mathcal{R}$ satisfies Y^D if and only if each strongly connected components (S.C.C.) of $GRD(\mathcal{R})$, except those composed of a single rule and no loop, satisfies Y.*

Proof Let $\mathcal{R}$ be a set of rules and Y be an acyclicity property. To ease the reading we use the notation from [30]: given an acyclicity property Y, a set of rules $\mathcal{R}$ satisfies $Y^{\prec}$ if all strongly connected components of $GRD(\mathcal{R})$ satisfy Y, except for those composed of a single rule and no loop. It should appear obvious that the lemma can be reformulated as $Y^D = Y^{\prec}$.

We first show that if $\mathcal{R}$ is not Y^D then it is not $Y^{\prec}$. Suppose that $\mathcal{R}$ does not satisfy Y^D. We then have an existential position $[a, i]$ in $PG^D(\mathcal{R})$ such that $[a, i] \in M([a, i])$, where M is the marking associated with Y. Specifically, this means that there is a cycle going through $[a, i]$ in $PG^D(\mathcal{R})$. Then all rules from this cycle belong to the same strongly connected component of $GRD(\mathcal{R})$. Consider the restriction of $\mathcal{R}$ to the set of rules $\mathcal{R}'$ that correspond to the S.C.C. in which the rules from this cycle appear. If we build $PG^F(\mathcal{R})$, we see that $\mathcal{R}'$ does not satisfy Y^F, hence Y. We have then exhibited a S.C.C. of the $GRD(\mathcal{R})$ that does not satisfy Y, hence $\mathcal{R}$ is not $Y^{\prec}$.

Now we show that if $\mathcal{R}$ is not $Y^{\prec}$, then it is not Y^D. Assume that $\mathcal{R}$ does not satisfy $Y^{\prec}$. Since it does not satisfy $Y^{\prec}$ there is at least one S.C.C. that does not satisfy Y. Call it $\mathcal{R}'$. Hence $PG^F(\mathcal{R}')$ contains an existential position $[a, i]$ belonging to a cycle. Since $\mathcal{R}$ (hence $\mathcal{R}'$) is Y^D, this cycle does not occur anymore in $PG^D(\mathcal{R}')$. However, the only edges we are allowed to remove in $PG^D(\mathcal{R}')$ are edges between rules R_i and R_j for which there is no path from R_i to R_j in $GRD(\mathcal{R})$. Thus, we cannot remove any edge (from the definition of a S.C.C.). Hence, $\mathcal{R}'$ is not Y^D. □

Proposition 10 *Let Y_1, Y_2 be two acyclicity properties. If $Y_1 \subseteq Y_2$, then $Y_1^D \subseteq Y_2^D$.*

Proof Consider a set of rules $\mathcal{R}$ that satisfies Y_1^D. From Lemma 1, each strongly connected component of $(^D\mathcal{R})$ satisfies Y_1. Since $Y_1 \subseteq Y_2$, each S.C.C. of $GRD(\mathcal{R})$ also satisfies Y_2, therefore $\mathcal{R}$ satisfies Y_2^D. □

Proposition 11 *Let Y be an acyclicity property. If $aGRD \not\subseteq Y$ then $Y \subset Y^D$.*

Proof Let $\mathcal{R}$ be a set of rules that does not satisfy Y but satisfies $aGRD$. From the definition of $aGRD$, $GRD(\mathcal{R})$ is composed of $|\mathcal{R}|$ strongly connected components with no loop. Thanks to Lemma 1, $\mathcal{R}$ trivially satisfies Y^D. Therefore, $\mathcal{R}$ is a set of rules satisfying Y^D but not Y. □

Proposition 12 *Let Y_1, Y_2 be two acyclicity properties such that $Y_1 \subset Y_2$, $wa \subseteq Y_1$ and $Y_2 \not\subseteq Y_1^D$. Then $Y_1^D \subset Y_2^D$.*

Proof Let $\mathcal{R}$ be a set of rules such that $\mathcal{R}$ satisfies Y_2 and neither Y_1 nor $aGRD$. $\mathcal{R}$ can be rewritten into $\mathcal{R}'$ by replacing each rule $R_i = H_i \leftarrow B_i \in \mathcal{R}$ with a new rule $R_i' = H_i \cup \{p(x)\} \leftarrow B_i \cup \{p(x)\}$ where p is a fresh predicate and x a fresh variable. Each rule can now be unified with each rule, but the only created cycles are those which contain only atoms $p(x)$, hence none of those cycles go through existential positions. Since $wa \subseteq Y_1$ (and so $wa \subseteq Y_2$), the added cycles do not change the behavior of $\mathcal{R}$ w.r.t. Y_1 and Y_2. Hence, $\mathcal{R}'$ is a set of rules satisfying Y_2 and not Y_1, and since $GRD(\mathcal{R}')$ is a complete graph, $\mathcal{PG}^D(\mathcal{R}') = \mathcal{PG}^F(\mathcal{R}')$. We can conclude that $\mathcal{R}'$ satisfies Y_2^D but not Y_1^D. □

Theorem 5 *Let Y be an acyclicity property. If $Y \subset Y^D$, then $Y^D \subset Y^U$. Furthermore, there is an injective mapping from the sets of rules satisfying Y^D but not Y, to the sets of rules satisfiying Y^U but not Y^D.*

Proof Assume $Y \subset Y^D$ and $\mathcal{R}$ satisfies Y^D but not Y. $\mathcal{R}$ can be rewritten into $\mathcal{R}'$ by applying the following steps. First, for each rule $R_i = H_i[\vec{y}, \vec{z}] \leftarrow B_i[\vec{x}, \vec{y}]. \in \mathcal{R}$, let $R_{i,1} = p_i(\vec{x}, \vec{y}) \leftarrow B_i[\vec{x}, \vec{y}]$. where p_i is a fresh predicate ; and $R_{i,2} = H_i[\vec{y}, \vec{z}] \leftarrow p_i(\vec{x}, \vec{y})$.. Then, for each rule $R_{i,1}$, let $R'_{i,1}$ be the rule $H_{i,1} \leftarrow B'_{i,1}$. with $B'_{i,1} = B_{i,1} \cup \{p'_{j,i}(x_{j,i}) : \forall R_j \in \mathcal{R}\}$, where $p'_{j,i}$ are fresh predicates and $x_{j,i}$ fresh variables. Now, for each rule $R_{i,2}$ let $R'_{i,2}$ be the rule $(B_{i,2} \leftarrow H'_{i,2}.)$ with $H'_{i,2} = H_{i,2} \cup \{p'_{i,j}(z_{i,j}) : \forall R_j \in \mathcal{R}\}$, where $z_{i,j}$ are fresh existential variables. Let $\mathcal{R}' = \bigcup_{R_i \in \mathcal{R}} \{R'_{i,1}, R'_{i,2}\}$. This construction ensures that each $R'_{i,2}$ depends on $R'_{i,1}$, and each $R'_{i,1}$ depends on each $R'_{j,2}$, thus, there is a *transition* edge from each $R'_{i,1}$ to $R'_{i,2}$ and from each $R'_{j,2}$ to each $R'_{i,1}$. Hence, $\mathcal{PG}^D(\mathcal{R}')$ contains exactly one cycle for each cycle in $\mathcal{PG}^F(\mathcal{R})$. Furthermore, $\mathcal{PG}^D(\mathcal{R}')$ contains at least one marked cycle w.r.t. Y, and then $\mathcal{R}'$ is not Y^D. Now, each cycle in $\mathcal{PG}^U(\mathcal{R}')$ is also a cycle in $\mathcal{PG}^D(\mathcal{R})$, and since $\mathcal{PG}^D(\mathcal{R})$ satisfies Y, $\mathcal{PG}^U(\mathcal{R}')$ also does. Hence, $\mathcal{R}'$ does not belong to Y^D but to Y^U. □

Theorem 6 *Let Y_1 and Y_2 be two acyclicity properties. If $Y_1^D \subset Y_2^D$ then $Y_1^U \subset Y_2^U$.*

Proof Let $\mathcal{R}$ be a set of rules such that $\mathcal{R}$ satisfies Y_2^D but does not satisfy Y_1^D. We rewrite $\mathcal{R}$ into $\mathcal{R}'$ by applying the following steps. For each pair of rules $R_i, R_j \in \mathcal{R}$ such that R_j depends on R_i, for each variable x in the frontier of R_j and each variable Y in the head of R_i, if x and Y occur both in a given predicate position, we add to the body of R_j a new atom $p_{i,j,X,Y}(X)$ and to the head of R_i a new atom $p_{i,j,X,Y}(Y)$, where $p_{i,j,X,Y}$ denotes a fresh predicate. This construction will allow each term from the head of R_i to propagate to each term from the body of R_j, if they shared some predicate position in $\mathcal{R}$. Thus, any cycle in $\mathcal{PG}^D(\mathcal{R})$ is also in $\mathcal{PG}^U(\mathcal{R}')$, without modifying behavior w.r.t. the acyclicity properties. Hence, $\mathcal{R}'$ satisfies Y_2^U but does not satisfy Y_1^U. □

Definition 22 (**Compatible unifier**) Let R_1 and R_2 be two rules. A unifier μ of B_2 with H_1 is compatible if, for each position $[a, i]$ in B'_2 (where B'_2 is the unified subset of B_2, see "dependency approach in Section 6) such that $\mu(term([a, i]))$ is an existential variable Z in H'_1, $PG^U(\mathcal{R})$ contains a path, from a position in which Z occurs, to $[a, i]$, that does not go through another existential position. Otherwise μ is incompatible.

Proposition 13 *Let R_1 and R_2 be two rules, and let μ be a unifier of B_2 with H_1. If μ is incompatible, then no application of R_2 can use an atom in $\mu(H_1)$. More formally, no* *application π' of R_2 can map an atom $a \in B_2$ to an atom b produced by an application (R_1, π) such that $b = \pi(b')$, where π and π' are more specific than μ.*

Proof Consider the application of R_1 to a set of facts F according to a homomorphism π' such that for an atom $a \in B_2$, $\pi'(a) = b = \pi(b')$, where both π and π' are more specific than μ. Note that this implies that $\mu(a) = \mu(b')$. Assume that b contains a fresh variable z_i produced from an existential variable z in H_1. Let z' be the variable from a such that $\pi'(z') = z_i$. Since the domain of π' is the variables of B_2, all atoms from B_2 in which z' occurs at a given position $[p, j]$ are also mapped by π' to atom containing z_i in the same position $[p, j]$. Since z_i is a fresh variable, these atoms have been produced by sequences of rule applications starting from (R_1, π). Such a sequence of rule applications exists only if there is a path in PG^U from a position of z in H_1 to $[p, j]$; moreover, this path cannot go through an existential position, otherwise z_i cannot be propagated. Hence μ is necessarily compatible. □

Definition 23 – Let R_1 and R_2 be rules such that there is a compatible unifier μ of B_2 with H_1. The associated unified rule $R_\mu = R_1 \diamond_\mu R_2$ is defined by $H_\mu = \mu(H_1) \cup \mu(H_2)$, and $B_\mu = \mu(B_1) \cup (\mu(B_2) \setminus \mu(H_1))$.
– Let $(R_1, \ldots, R_{k+1})$ be a sequence of rules. A sequence $s = (R_1\mu_1R_2 \ldots \mu_kR_{k+1})$, where for $1 \leq i \leq k$, μ_i is a unifier of B_{i+1} with H_i, is a compatible sequence of unifiers if :
  - μ_1 is a compatible unifier of B_2 with H_1;
  - if $k > 0$, the sequence obtained from s by replacing $(R_1\mu_1R_2)$ with $R_1 \diamond_{\mu_1} R_2$ is a compatible sequence of unifiers.

Definition 24 (**Compatible cycles**) Let Y be an acyclicity property, and PG^U be a position graph with unifiers. The compatible cycles for $[a, i]$ in PG^U are all marked cycles C for $[a, i]$ w.r.t. Y, such that there is a compatible sequence of unifiers induced by C. Property Y^{U+} is satisfied if, for each existential position $[a, i]$, there is no compatible cycle for $[a, i]$ in PG^U.

Proposition 14 *Let Y be an acylicity property. Then, $Y^U \subseteq Y^{U+}$. Moreover, if $Y^D \subset Y^U$ then $Y^U \subset Y^{U+}$.*

Proof Inclusion follows immediately from the definitions.

We now show that this inclusion is strict. Let $\mathcal{R}$ be a set of rules satisfying Y^U but not Y^D. We build a set of rules $\mathcal{R}'$ that satisfies Y^{U+} but not Y^U. To this aim, we first increase the arity of each predicate of $\mathcal{R}$ by two, and in each rule body and head, we put two fresh variables t_1 and t_2 in those positions. E.g., a rule $s(x, y) \rightarrow t(y, z)$ would become $s(x, y, t_1, t_2) \rightarrow t(y, z, t_1, t_2)$. Then, for each rule $R = (B, H)$, we create four fresh predicates p, q_1, q_2, r whose arity is respectively $|var(H)|$, 2, 2 and 2, and five fresh variables z_1, z_2, z_3, z_4 and z_5. Then we "split" R into four rules (where $\vec{x}$ is a list of all variables from H):

- $R_1 = B \rightarrow p(\vec{x}, z_1, z_2)$,
- $R_2 = p(\vec{x}, z_1, z_2) \rightarrow q_1(z_1, z_3)$,
- $R_3 = q_1(z_1, z_3) \rightarrow s(z_3, z_5)$,
- $R_4 = p(\vec{x}, z_1, z_2) \wedge q_1(z_1, z_3) \wedge q_2(z_1, z_4) \wedge s(z_3, z_5) \wedge s(z_4, z_5) \rightarrow H$.

The graph of rule dependencies of those four rules contains the following edges: (R_1, R_2), (R_2, R_3), (R_3, R_4). It can be observed that in particular, in $PG^U(\mathcal{R}')$ there is a transition edge going from the last position of the atom $p(\vec{x}, z_1, z_2)$ in rule R_1 to the last position of the "same" atom in rule R_4. The same holds for the penultimate position of these atoms. However, it can be seen that given any set of facts, rule R_4 can never be applied. But the definition of PG^U does not take this "complicated" interactions into account. Specifically, the set of rules is not Y^U anymore.

Let us now consider Y^{U+}. There is no compatible cycle in PG^U since the existential variable z_1 in rule R_1 has to go through new existential positions before reaching the position of z_1 in rule R_4. Thus, $\mathcal{R}'$ is Y^{U+}. □

Proposition 15 *Let Y_1 and Y_2 be two acyclicity properties. If $Y_1^D \subset Y_2^D$, and $Y_2^D \subset Y_2^{U+}$, then $Y_1^{U+} \subset Y_2^{U+}$.*

Proof Observe that the transformation we used in the proof of Theorem 6 actually guarantees that all cycles which are present are compatible cycles. Thus, for the obtained set of rules $\mathcal{R}'$ and any acyclicity property Y, $\mathcal{R}'$ satisfies Y^U if and only if $\mathcal{R}'$ satisfies Y^{U+}. □

Theorem 7 *Let Y be an acyclicity property ensuring the halting the chase. Then, the chase halts for any set of rules $\mathcal{R}$ that satisfies Y^{U+} (hence Y^U and Y^D).*

Proof (sketch) The complete proof is technically involved, and the reader is referred to [44] for more details. The idea is that if the chase does not halt, then there exists some existential position which is infinitely often populated by new individuals. Such a position must occur in some cycle in PG^U, as our construction only "removes" edges that do not correspond to "real" rule applications. Furthermore, Proposition 13 ensures that the cycle cannot be ignored by Y^{U+}. □

Theorem 8 (**Complexity of Recognition**) *Let Y be an acyclity property, and $\mathcal{R}$ be a set of rules. If checking that $\mathcal{R}$ satisfies Y is in coNP, then, checking that $\mathcal{R}$ satisfies Y^D, Y^U or Y^{U+} is coNP-complete.*

Proof One can guess a cycle in $PG^D(\mathcal{R})$ (or $PG^U(\mathcal{R})$, or $PG^{U+}(\mathcal{R})$) such that the property Y is not satisfied by this cycle. Each edge of the cycle has a polynomial certificate, since checking if a given substitution is a piece-unifier can be done in polynomial time. Since Y is in $coNP$, we have a polynomial certificate that this cycle does not satisfy Y. Membership in $coNP$ follows.

The completeness part is proved by a simple reduction from the co-problem of rule dependency checking (which is thus a coNP-complete problem). Rule dependency checking is equivalent to finding an atom-erasing unifier (see "the dependency approach" in Section 6). Let R_1 and R_2 be two rules. We first define two fresh predicates p and s of arity $|var(B_1)|$ and two fresh predicates q and r of arity $|var(H_2)|$. We build $R_0 = p(\vec{x}) \rightarrow s(\vec{x})$ where $\vec{x}$ is a list of all variables in B_1, and $R_3 = r(\vec{x}) \rightarrow p(\vec{z}) \wedge q(\vec{x})$, where $\vec{z} = (z, z, \ldots, z)$, where z is a variable which does not appear in H_2. We rewrite R_1 into $R'_1 = B_1 \wedge s(\vec{x}) \rightarrow H_1$ and R_2 into $R'_2 = B_2 \rightarrow H_2 \wedge r(\vec{x})$, where $\vec{x}$ is a list of all variables in H_2. One can check that $\mathcal{R} = \{R_0, R'_1, R'_2, R_3\}$ contains a cycle going through an existential variable (thus, it is not wa^D) iff R_2 depends on R_1. □

7 Termination of ASPeRiX computations

Consider P an ENM-program. In Section 3, we have defined the semantics of this program as the semantics of the partial grounding of its skolemization. In an ASPeRiX computation of this program, the IN fields generated thus correspond to a skolem-derivation using the rules in *pos*(P) (i.e., the existential rules obtained by removing negative bodies from all rules in P). It is easy to check that:

Proposition 16 *Let P be an ENM-program. If the Skolem chase halts on pos(P) then, the ASPeRiX computation halts on P.*

This proposition allows us to use all decidability results presented in Section 6, since all those decidable classes halt with the Skolem chase.

We have seen in Section 5 that some chases were stronger than the Skolem chase, and could halt where the Skolem chase couldn't. An immediate question is "what happens if we replace the Skolem chase used in the ASPeRiX computation by some other C-chase, thus defining an *ASPeRiX C-computation*?"

We first show that those different algorithms produce different results, and thus implement different semantics. These semantics are discussed in Section 7.1. Then we show in Section 7.2 that Proposition 16 does not extend easily to other computations. Finally, in Section 7.3, we provide a sufficient condition on negative bodies ensuring termination of ASPeRiX computations.

7.1 Semantics of ASPeRiX C-computations

In the positive case, all chase variants produce equivalent universal models (up to skolemization). Moreover, running a chase on equivalent knowledge bases produce equivalent results. Do these semantic properties still hold with nonmonotonic existential rules? The answer is no in general.

The next example shows that the chase variants presented in this paper, core chase excepted, may produce non-equivalent results from equivalent knowledge bases.

Example 19 Let $F = \{p(a, Y), t(Y)\}$ and $F' = \{p(a, Y'), p(a, Y), t(Y)\}$ be two equivalent atomsets. Let $R : r(U) \leftarrow p(U, V), not\ t(V)$.. For any ASPeRiX C-computation other than core chase, there is a single result for $(F, \{R\})$ which is F (or *sk(F)*) and a single result for $(F', \{R\})$ which is $F' \cup \{r(a)\}$ (or $sk(F') \cup \{r(a)\}$). These sets are not equivalent.

Of course, if we consider that the initial knowledge base is already skolemized (including F seen as a rule), this trouble does not occur with the Skolem-chase since there are no redundancies in facts and no redundancy can be created by a rule application. This problem does not arise with core chase either. Thus the only two candidates for processing ENM-rules are the core chase and the Skolem chase (if we assume *a priori* skolemisation, which is already a semantic shift).

On the one hand, the core chase is more expensive (since at each step of the breadth-first forward chaining there is a redundancy check possibly accompanied by the computation of a core, which can be done with a number of homomorphism checks linear in the number of facts). On the other hand, the core chase allows to keep the original knowledge base and terminates more often than the Skolem chase.

The choice between both mechanisms is important since, as shown by the next example, they may produce different results even when they both produce a *unique* result. It follows that skolemizing existential rules is not an innocuous transformation in presence of nonmontonic negation.

Example 20 We consider $F = i(a)$, $R_1 = p(X, Y) \leftarrow i(X).$, $R_2 = q(X, Y) \leftarrow i(X).$, $R_3 = p(X, Y), t(Y) \leftarrow q(X, Y).$ and $R_4 = r(U) \leftarrow p(U, V), not\ t(V)..$ The core chase produces at first step $p(a, Y_0)$ and $q(a, Y_1)$, then $p(a, Y_1)$ and $t(Y_1)$ and removes the redundant atom $p(a, Y_0)$, hence R_4 is not applicable. The unique result of the ASPeRiX core-computation is $\{i(a), q(a, Y_1), p(a, Y_1), t(Y_1)\}$. With the Skolem chase, the produced atoms are $p(a, f^{R_1}(a))$ and $q(a, f^{R_2}(a))$, then $p(a, f^{R_2}(a))$ and $t(f^{R_2}(a))$. R_4 is applied with $p(U, V)$ mapped to $p(a, f^{R_1}(a))$, which produces $r(a)$. These atoms yield a unique ASPeRiX Skolem-computation result. These results are not equivalent.

The relationships between both kinds of chase applied to nonmonotonic existential rules can be specified as follows: *(1)* For result S of the ASPeRiX core-computation, there is a result S' of the ASPeRiX Skolem-computation with an homomorphism from S to S'; *(2)* the ASPeRiX Skolem-computation may produce strictly more results than the ASPeRiX core-computation, even infinitely many more.

7.2 Termination of ASPeRiX C-computations

We say that the ASPeRiX C-halts on $(F, \mathcal{R})$ when there exists a finite ASPeRiX C-computation of $(F, \mathcal{R})$ (in that case, a breadth-first strategy for the rule applications will generate it). We can thus define *C-ENM-finite* as the class of sets of nonmonotonic existential rules $\mathcal{R}$ for which ASPeRiX C-halts on any $(F, \mathcal{R})$. Our first intuition was to assert "if $pos(\mathcal{R}) \in C$-finite, then $\mathcal{R} \in C$-ENM-finite". However, this property is not true in general, as shown by the following example:

Example 21 Let $\mathcal{R} = \{R_1, R_2\}$ where $R_1 = p(X, Y), h(Y) \leftarrow h(X).$ and $R_2 = p(X, X) \leftarrow p(X, Y), not\ h(X)..$ See that $pos(\mathcal{R}) \in$ core-finite (as soon as R_1 is applied, R_2 is also applied and the loop $p(X, X)$ makes any other rule application redundant); however the only result of an ASPeRiX core-computation of $(\{h(a)\}, \mathcal{R})$ is infinite (because all applications of R_2 are blocked).

The following property shows that the desired property is true for *local* chases.

Proposition 17 *Let $\mathcal{R}$ be a set of ENM-rules and C be a local chase. If $pos(\mathcal{R}) \in C$-finite, then $\mathcal{R} \in C$-ENM-finite.*

We have previously argued that the only two interesting chase variants w.r.t. the desired semantic properties are Skolem and core. However, the core-finiteness of the positive part of a set of ENM-rules does not ensure the core-stable-finiteness of these rules. We should point out now that if $C \geq C'$, then C'-ENM-finiteness implies C-ENM-finiteness. We can thus ensure core-ENM-finiteness when C-finiteness of the positive part of rules is ensured for a local C-chase.

Proposition 18 *Let $\mathcal{R}$ be a set of ENM-rules and C be a local chase. If pos($\mathcal{R}$) $\in$ C-finite, then $\mathcal{R} \in$ core-ENM-finite.*

We can thus rely upon all acyclicity results in this paper (for which the Skolem chase halts) to ensure that the ASPeRiX core-computation also halts.

7.3 Using negative bodies to ensure termination

We now explain how negation can be exploited to enhance all previous acyclicity notions. We first define the notion of *self-blocking rule*, which is a rule that will never be applied in any derivation.

Definition 25 (**Self-blocking rule**) Let $R : H \leftarrow B^+, B_1^-, \ldots B_k^-$ be an ENM-rule. R is self-blocking if there is a negative body B_i^- such that $B_i^- \subseteq B^+ \cup H$.

Such a rule will never be applied in a sound way, so will never produce any atom. It follows that:

Proposition 19 *Let $\mathcal{R}'$ be the non-self-blocking rules of $\mathcal{R}$. If pos($\mathcal{R}'$) $\in$ C-finite and C is local, then $\mathcal{R} \in$ C-ENM-finite.*

This idea can be further extended. We have seen for existential rules that if $R' : H' \leftarrow B'$ depends on $R : H \leftarrow B$, then there is a unifier μ of B' with H, and we can build a rule $R'' = R \diamond_\mu R'$ that captures the sequence of applications encoded by the unifier. We extend Definition 23 to take into account negative bodies: if B^- is a negative body of R or R', then $\mu(B^-)$ is a negative body of R''. We also extend the notion of dependency in a natural way, and say that a unifier μ of B' with H is self-blocking when $R \diamond_\mu R'$ is self-blocking, and R' *depends* on R when there exists a unifier of B' with H that is not self-blocking. This extended notion of dependency exactly corresponds to the *positive reliance* in [40].

Example 22 Let $R = r(X, Y) \leftarrow q(X), not\ \ p(X).$ and $R' = p(X), q(Y) \leftarrow r(X, Y)..$ Their associated positive rules are not core-finite. There is a single unifier μ of R' with R, and $R \diamond_\mu R' : r(X, Y), p(X), q(Y) \leftarrow q(X), not\ \ p(X).$ is self-blocking. Then the Skolem-chase-tree halts on $(F, \{R, R'\})$ for any F.

Results obtained from positive rules can thus be generalized by considering this extended notion of dependency (for $\mathcal{PG}^U$ we only encode non self-blocking unifiers). Note that it does not change the complexity of the acyclicity tests.

We can further generalize this and check if a unifier sequence is self-blocking, thus extend the Y^{U+} classes to take into account negative bodies. Let us consider a compatible cycle C going through $[a, i]$ that has not been proven safe. Let C_μ be the set of all compatible unifier sequences induced by C. We say that a sequence $\mu_1 \ldots \mu_k \in \mathcal{C}_\mu$ is self-blocking when the rule $R_1 \diamond_{\mu_1} R_2 \dots R_k \diamond_{\mu_k} R_{k+1}$ obtained by combining these unifiers is self-blocking. When all sequences in C_μ are self-blocking, we say that C is also self-blocking. This test comes again at no additional computational cost.

Example 23 Let $R_1 = r(X_1, Y_1) \leftarrow q(X_1), not\ p(X_1).$, $R_2 = s(X_2, Y_2) \leftarrow r(X_2, Y_2).$, $R_3 = p(X_3), q(Y_3) \leftarrow s(X_3, Y_3).$. $PG^{U+}(\{R_1, R_2, R_3\})$ has a unique cycle, with a unique induced compatible unifier sequence. The rule $R_1 \diamond R_2 \diamond R_3 = r(X_1, Y_1), s(X_1, Y_1), p(X_1), q(Y_1) \leftarrow q(X_1), not\ p(X_1).$ is self-blocking, hence $R_1 \diamond R_2 \diamond R_3 \diamond R_1$ also is. Thus, there is no "dangerous" cycle.

Proposition 20 *Let $\mathcal{R}$ be a set of* ENM*-rules. If, for each existential position $[a, i]$ in a rule in $\mathcal{R}$, all compatible cycles for $[a, i]$ in $\mathcal{PG}^U$ are self-blocking, then the* ASPeRiX *Skolem-computation halts on $\mathcal{R}$.*

8 Conclusion

This paper has presented a new formalism called *existential non-monotonic rules (*ENM*-rules)* which integrates ontologies and rules in a unique formalism and offers a computational study of this formalism. On one hand, it expands the standard ASP formalism by allowing the use of existential variables. On the other hand, it expands the standard existential rules formalism by allowing the use of default negation. From a practical point of view, the proposed translation from ENM-rules to ASP allows us to use any solvers. But let us note that we have implemented this translation as a front-end of the solver ASPeRiX which uses on-the-fly grounding [36]. This should help, in the future, for dealing with variables in a more efficient way.

Compared to other approaches, the present work has the following advantages: it uses a unique formalism and a unique semantics for ontologies and rules; it does not suffer from the important restrictions sometimes imposed, such as stratified negation; and it is actually implemented.

Moreover, we have revisited chase termination for existential rules with several results. First, we have presented a new tool that allows to unify and extend most existing acyclicity conditions, while keeping good computational properties. Second, we have discussed a chase-like mechanism for ENM-rules, and the extension of acyclicity conditions to take negation into account.

The main ongoing work consists in dealing 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 but it is a promising way when dealing with ontological information issued from the web.

Acknowledgments This work received support from ANR (French National Research Agency), ASPIQ project reference ANR-12-BS02-0003. Q4

References

1. Ahmetaj, S., Ortiz, M., Simkus, M.: Polynomial datalog rewritings for expressive description logics with closed predicates. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, pp. 878–885. New York (2016)

2. Alviano, M., Morak, M., Pieris, A.: Stable model semantics for tuple-generating dependencies revisited. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, pp. 377–388. Chicago (2017)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the el envelope. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05, pp. 364–369. Morgan Kaufmann Publishers Inc, San Francisco (2005)
4. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: On rules with existential variables: walking the decidability line. Artif. Intell. **175**(9–10), 1620–1654 (2011)
5. Baget, J.-F., Mugnier, M.-L.: The complexity of rules and constraints. J. Artif. Intell. Res. (JAIR) **16**, 425–465 (2002)
6. Baget, J.-F.: Ontologies and large databases: Querying algorithms for the Web of Data. Invited Talk, Artificial Intelligence meets the Web of Data. ESWC'13 Workshop (2013)
7. Baget, J.-F.: Improving the forward chaining algorithm for conceptual graphs rules. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), pp. 407–414. Whistler (2004)
8. Baget, J.-F., Garreau, F., Mugnier, M.-L., Rocher, S.: Extending acyclicity notions for existential rules. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, pp 39–44. Prague (2014)
9. Baget, J.-F., Garreau, F., Mugnier, M.-L., Rocher, S.: Revisiting chase termination for existential rules and their extension to nonmonotonic negation. In: Konieczny, S., Tompits, H. (eds.) NMR'2014: 15th International Workshop on Non-Monotonic Reasoning, Volume INFSYS Research Report Series, Vienna (2014)
10. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: Extending decidable cases for rules with existential variables. In: IJCAI'09: 21st International Joint Conference on Artificial Intelligence, pp. 677–682. AAAI, Pasadena (2009)
11. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
12. Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: Automata, Languages and Programming, 8th Colloquium, Acre (Akko). Proceedings, pp. 73–85. Israel (1981)
13. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. In: KR'08, pp. 70–80 (2008)
14. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. In: PODS'09, pp. 77–86 (2009)
15. Calì, A., Gottlob, G., Lukasiewicz, T.: Tractable query answering over ontologies with datalog+/-. In: Proceedings of the 22nd International Workshop on Description Logics (DL-2009) (2009)
16. Calimeri, F., Cozza, S., Ianni, G., Leone, N.: Computable functions in ASP: Theory and implementation. In: Logic Programming, 24th International Conference, ICLP 2008. Proceedings, pp. 407–424. Udine (2008)
17. Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. J. Autom. Reas. **39**(3), 385–429 (2007)
18. Chandra, A.K., Lewis, H.R., Makowsky, J.A.: Embedded implicational dependencies and their inference problem. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing, pp. 342–354. Milwaukee (1981)
19. de Bruijn, J., Pearce, D., Polleres, A., Valverde, A.: A semantical framework for hybrid knowledge bases. Knowl. Inf. Syst. **25**(1), 81–104 (2010)
20. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: PODS'08, pp. 149–158 (2008)
21. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. **172**(12–13), 1495–1539 (2008)
22. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comput. Sci. **336**(1), 89–124 (2005)
23. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. In: Database Theory - ICDT 2003, 9th International Conference. Proceedings, pp. 207–224. Siena (2003)
24. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artif. Intell. **175**, 236–263 (2011)
25. Garreau, F., Garcia, L., Lefèvre, C., Stéphan, I.: ∃-asp. In: Proceedings of the Ontologies and Logic Programming for Query Answering workshop (ONTOLP'15). Buenos Aires (2015)
26. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K. (eds.) Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP'88), pp. 1070–1080. The MIT Press, Cambridge (1988)
27. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. Gen Comput **9**(3/4), 365–386 (1991)

28. Gottlob, G., Hernich, A., Kupke, C., Lukasiewicz, T.: Equality-friendly well-founded semantics and applications to description logics. In: Hoffmann, J., Selman, B. (eds.) Proceedings of the 26th National Conference on Artificial Intelligence, AAAI 2012. AAAI Press, Toronto (2012)
29. Grau, B.C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity conditions and their application to query answering in description logics. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012. AAAI Press, Rome (2012)
30. Grau, B.C., Horrocks, I., Kròtzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity notions for existential rules and their application to query answering in ontologies. J. Artif. Intell. Res. (JAIR) **47**, 741–808 (2013)
31. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Open answer set programming for the semantic web. J. Appl. Logic **5**(1), 144–169 (2007). Questions and Answers: Theoretical and Applied Perspectives
32. Ianni, G., Eiter, T., Tompits, H., Schindlauer, R.: Nlp-dl: A kr system for coupling nonmonotonic logic programs with description logics. In: The Forth International Semantic Web Conference (ISWC2005) (2005)
33. Kròtzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, pp. 963–968. Barcelona (2011)
34. Lee, J., Palla, R.: Integrating rules and ontologies in the first-order stable model semantics (preliminary report). In: Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011. Proceedings, pp. 248–253. Vancouver (2011)
35. Lefèvre, C., Nicolas, P.: A first order forward chaining approach for answer set computing. In: Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09), volume 5753 of LNCS, pp. 196–208. Springer (2009)
36. Lefèvre, C., Bèatrix, C., Stèphan, I., Garcia, L.: Asperix, a first order forward chaining approach for answer set computing. CoRR, arXiv:1503.07717 :(to appear in TPLP) (2015)
37. Leone, N., Manna, M., Terracina, G., Veltri, P.: Efficiently computable datalog$^\exists$ programs. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012. Rome (2012)
38. Lierler, Y., Lifschitz, V.: One more decidable class of finitely ground programs. In: Logic Programming, 25th International Conference, ICLP 2009. Proceedings, pp. 489–493. Pasadena (2009)
39. Liu, L., Pontelli, E., Son, T.C., Truszczynski, M.: Logic programs with abstract constraint atoms: The role of computations. Artif. Intell. **174**(3-4), 295–315 (2010)
40. Magka, D., Kròtzsch, M., Horrocks, I.: Computing stable models for nonmonotonic existential rules. In: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence. Beijing (2013)
41. Marnette, B.: Generalized schema-mappings: From termination to tractability. In: PODS, pp. 13–22 (2009)
42. Motik, B., Rosati, R.: Reconciling description logics and rules. J. ACM, **57**(5) (2010)
43. Mugnier, M.-L.: Ontological query answering with existential rules. In: Web Reasoning and Rule Systems - 5th International Conference, RR 2011Proceedings, pp. 2–23. Galway (2011)
44. Rocher, S.: Querying Existential Rule Knowledge Bases: Decidability and Complexity. PhD thesis, Université de Montpellier, France (2016)
45. Rosati, R.: Dl+log: Tight integration of description logics and disjunctive datalog. In: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, pp. 68–78. Lake District of the United Kingdom (2006)
46. Salvat, E., Mugnier, M.-L.: Sound and complete forward and backward chainingd of graph rules. In: Eklund, P.W., Ellis, G., Mann, G. (eds.) Conceptual Structures: Knowledge Representation as Interlingua, 4th International Conference on Conceptual Structures, ICCS '96. Proceedings, Volume 1115 of Lecture Notes in Computer Science, pp. 248–262. Springer, Sydney (1996)
47. Wan, H., Zhang, H., Xiao, P., Huang, H., Zhang, Y.: Query answering with inconsistent existential rules under stable model semantics. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 1095–1101. AAAI Press, Phoenix (2016)

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES:

Q1. Please check affiliations if captured and presented correctly.
Q2. Mathematics Subject Classification (2010) is required. Please provide.
Q3. Missing citation for Figure 1 was inserted here. Please check if appropriate. Otherwise, please provide citation for Figure 1. Note that the order of main citations of figures/tables in the text must be sequential.
Q4. Please check Acknowledgments if captured and presented correctly.