

\exists -ASP for Computing Repairs with Existential Ontologies*

Jean-François Baget¹, Zied Bouraoui², Farid Nouioua³, Odile Papini³, Swan Rocher⁴,
and Eric Würbel³

¹ INRIA Montpellier, France - baget@lirmm.fr

² Cardiff University, UK - BouraouiZ@cardiff.ac.uk

³ Aix-Marseille Univ, France - {farid.nouioua,odile.papini,eric.wurbel}@univ-amu.fr

⁴ Univ Montpellier, France - rocher@lirmm.fr

Abstract. Repair-based techniques are a standard way of dealing with inconsistency in the context of ontology-based data access where several inconsistency-tolerant semantics have been mainly proposed for lightweight description logics. In this paper we present a generic transformation from knowledge bases expressed within existential rules formalism into an ASP program. We propose different strategies for this transformation, and highlight the ones for which answer sets of the generated program correspond to various kinds of repairs used in inconsistency-tolerant inferences.

1 Introduction

Dealing with inconsistency in ontology-based query answering is one of the challenging problems that received a lot of attention in recent years, (e.g. [2,7,11,17]). In such a setting, inconsistency problem comes from the data, i.e. occurs when assertional facts contradict constraints imposed by the ontological knowledge. In case of inconsistency, standard inference is meaningless: All queries would be positively answered. In this paper we focus on the mainstream approach that considers that the ontology, built by experts, is correct, and that only data has to be repaired. Other approaches (e.g. [20]) rely upon the assumption that the database is reliable but the rules are not. The latter assumption will not be explored in this paper and left for future work.

Many works (e.g. [12,16,18]), basically inspired by the approaches proposed in database area (e.g. [1,9]) and in propositional logic (e.g. [8]), deal with inconsistency by proposing several inconsistency-tolerant inferences, called *semantics*. These semantics are based on the notion of assertional base repair which is closely related to the notion of database repair [16] or maximally consistent subbase used in the propositional logic setting. An ABox repair is simply an assertional subbase which is consistent with an ontology. Ontology-based consistent query answering (AR-semantics) [16] comes down first to compute the set of repairs (i.e. all possible maximally consistent subsets of facts consistent with the ontological knowledge) and then to check to which extent a query can be entailed using these repairs. As shown in [16,10], the AR-semantics (also called

* This work was supported by the projects ASPIQ (ANR-12-BS02-0003) and the ERC Starting Grant 637277.

universal entailment) is a hard task (co-NP complete) for lightweight DLs [16,19]. In fact, inconsistency-tolerant semantics were introduced for the lightweight description logics *DL-Lite* (e.g. [16]), and later extended to other description logics (e.g. [19]) or existential rules (e.g. [17]). In this paper, we use existential rules (e.g. [5]) (also called Datalog+/-) as ontology language that generalizes lightweight description logics, such as *DL-Lite* and *EL* by allowing the use of any predicate arity as well as cyclic structures.

Recently the ASP framework [6], a convenient paradigm for knowledge representation and reasoning, especially when information is incomplete, has been enriched in order to deal with existential variables [13]. \exists -ASP is a fragment of ASP that generalises skolemized existential rules. It allows for enriching lightweight description logics with non-monotonic features, and benefits from decidability results obtained for existential rules. \exists -ASP has been naturally implemented on top of the ASP solver ASPeRiX¹, which does not rely on preliminary grounding to compute answer sets [15,14].

The paper first recalls the logical frameworks used in this paper: Existential rules in Section 2, \exists -ASP in Section 3, and the best known notions of repair in Section 4. Our contribution is presented in Section 5. We present a generic transformation from knowledge bases expressed within existential rules formalism into an ASP program. We propose different strategies for this transformation, and highlight the ones for which answer sets of the generated program correspond to various kinds of repairs used in inconsistency-tolerant inferences. The sound and complete \exists -ASP algorithm which is central to ASPeRiX computations will be used to prove the one-to-one correspondence between the answer sets of the generated program and the knowledge base repairs.

2 Existential Rules

We consider a *vocabulary* \mathcal{V} consisting of three disjoint sets, the set \mathcal{P} of *predicate names*, the set \mathcal{F} of *function symbols* (each provided with an arity) and the set \mathcal{C} of *constants*. Disjoint with \mathcal{V} , we also consider a set \mathcal{X} of *variables*. In what follows, constants will be notated in lowercase and variables in uppercase. The set of *terms* is defined inductively as follows: Constants and variables alike are terms, and if $f \in \mathcal{F}$ is a function symbol of arity k and t_1, \dots, t_p are terms, then $f(t_1, \dots, t_p)$ is also a term. An *atom* is an object of form $p(t_1, \dots, t_k)$, where p is a predicate name of arity k and the t_i are *terms*. An atom is said *basic* when none of its terms involve any function symbol, and is said *grounded* when no variable is used to define any of its terms. A set of atoms is said basic (resp. grounded) when all its atoms are basic (resp. grounded).

Homomorphisms A *substitution* is a mapping σ from a set of variables to a set of terms. If A is a set of atoms and σ is a substitution, we note $\sigma(A)$ the set of atoms obtained, for each variable x appearing both in an atom of A and the domain of σ , by replacing non-recursively each occurrence of x in A by $\sigma(x)$. For example, let $A = \{p(f(X, Y), Z), q(X, a)\}$ and $\sigma : X \mapsto f(X, a), Y \mapsto X$. Then $\sigma(A) = \{p(f(f(X, a), X), Z), q(f(X, a), a)\}$. Let F and Q be two sets of atoms. A *homomorphism* from Q to F is a substitution σ such that $\sigma(Q) \subseteq F$. If we note $\phi(A)$ the first-order logics (FOL) formula obtained by the conjunction of the atoms in A , and $\Phi(A)$

¹ available at <http://www.info.univ-angers.fr/pub/claire/asperix/>

the existential closure of $\phi(A)$, it is well known that $\Phi(F) \models \Phi(Q)$ iff there exists a homomorphism from Q to F . Let σ be a bijective substitution from the variables of F to a fresh set of constants (that appear neither in F nor in Q). The ground set of atoms $\sigma(F)$ is called a *grounding* of F and it holds that $\Phi(F) \models \Phi(Q)$ iff $\Phi(\sigma(F)) \models \Phi(Q)$.

Existential Rules An *existential rule* is of form $B \rightarrow H$ where both the *body* B and the *head* H are sets of *basic* atoms. We often note such a rule $B[\mathbf{X}, \mathbf{Y}] \rightarrow H[\mathbf{Y}, \mathbf{Z}]$, where the variables in \mathbf{X} are those that appear only in the body, the variables in \mathbf{Y} (called the *frontier*) are those that appear both in the body and the head, and those in \mathbf{Z} (called *existential variables*) are those that appear only in the head. The FOL formula associated with this existential rule is $\forall \mathbf{X} \forall \mathbf{Y} (\phi(B) \rightarrow (\exists \mathbf{Z} \phi(H)))$. For example, the FOL formula associated with $p(X, Y), r(X, Y', a) \rightarrow r(Y, Y', Z), p(Z, Z')$ is $\forall X \forall Y \forall Y' (p(X, Y) \wedge r(X, Y', a) \rightarrow (\exists Z \exists Z' r(Y, Y', Z) \wedge p(Z, Z')))$. Let $R = B \rightarrow H$ be a rule with frontier \mathbf{Y} and existential variables \mathbf{Z} . Let us consider a substitution σ_R that maps each existential variable $Z \in \mathbf{Z}$ to a functional term $f_z^R(\mathbf{Y})$. Then we say that $sk(R) = B \rightarrow \sigma_R(H)$ is a *skolemization* of R . Let R be the rule given in the example, then $sk(R) = p(X, Y), r(X, Y', a) \rightarrow r(Y, Y', f_Z^R(Y, Y')), p(f_Z^R(Y, Y'), f_{Z'}^R(Y, Y'))$.

Derivations Consider now a set of atoms F and a skolemized existential rule $R = B \rightarrow H$. We say that R is *applicable* to F when there exists a homomorphism σ from B to F . In that case, the *application of R on F according to σ* produces a set of atoms $\alpha(F, R, \sigma) = F \cup \sigma(H)$. Note that when F is ground, $\alpha(F, R, \sigma)$ is also ground. Let R be the rule given in the previous example, and $F = p(a, g(b)), r(a, g(b), a)$. The substitution $\sigma : X \mapsto a, Y \mapsto g(b), Y' \mapsto g(b)$ is a homomorphism from B to F and $\alpha(F, R, \sigma) = F \cup \{r(g(b), g(b), f_Z^R(g(b), g(b))), p(f_Z^R(g(b), g(b)), f_{Z'}^R(g(b), g(b)))\}$.

Let F be a set of atoms and \mathcal{R} be a set of rules. A \mathcal{R} -*derivation* from F is a (possibly infinite) sequence $F = F_0, F_1, \dots, F_i, \dots$ such that, $\forall i > 0$, there exists a rule $R = B \rightarrow H \in \mathcal{R}$ and a homomorphism σ from B to F_{i-1} such that $F_i = \alpha(F_{i-1}, R, \sigma)$. The *result* of a finite derivation F_0, \dots, F_k is the set of atoms F_k , when it is infinite we define it as the (infinite) union of all F_i . A derivation is said *full* when, for every rule $R = B \rightarrow H \in \mathcal{R}$, for every homomorphism σ from B to its result, there exists some F_i in the derivation such that $F_{i+1} = \alpha(F_i, R, \sigma)$. Any full \mathcal{R} -derivation on F produces the same result, and we call that result the \mathcal{R} -closure of F and note it $Cl_{\mathcal{R}}(F)$ (or simply $Cl(F)$ when there is no ambiguity on \mathcal{R}). When we consider a set $\Pi = F \cup \mathcal{R}$ as a program (as in Section 3), we note $Cl(\Pi) = Cl_{\mathcal{R}}(F)$.

Theorem 1. *Let F and Q be two set of atoms, and \mathcal{R} be a set of existential rules. We note F_g a grounding of F and \mathcal{R}_{sk} the skolemization of \mathcal{R} . Then $Cl_{\mathcal{R}_{sk}}(F_g)$ is a universal model, i.e., $F, \mathcal{R} \models Q$ iff there is a homomorphism from Q to $Cl_{\mathcal{R}_{sk}}(F_g)$.*

Skolem Chase Deciding whether or not $F, \mathcal{R} \models Q$ is undecidable. However, for all positive instances of the problem, a homomorphism from Q to $Cl_{\mathcal{R}_{sk}}(F_g)$ can be found after finitely many steps of a breadth first derivation. Such a derivation is called the *skolem chase*. For a more precise relationship between the skolem chase and other chases found in the literature, the reader can refer to [4]. A lot of work has been devoted to predicting that the chase will stop. Acyclicity conditions on a set of existential rules such as the ones presented in [3] ensure that the closure $Cl_{\mathcal{R}_{sk}}(F_g)$ will be finite.

3 Existential ASP

Syntax An *existential ASP* (\exists -ASP) rule is of form $H \leftarrow B^+, \text{not } B_1^-, \dots, \text{not } B_k^-$ where the *positive body* B^+ , the *negative bodies* B^- and the *head* H are sets of basic atoms. Intuitively, such a rule means “if the positive body is verified, and none of the negative bodies are, then we can conclude with the head”. To make our definitions easier to read, and without loss of generality (see the safety condition in [13]), we consider that all variables appearing in negative bodies also appear in the positive body. An \exists -ASP program is a set Π_F of basic atoms and a set $\Pi_{\mathcal{R}}$ of \exists -ASP rules. As for existential rules, we can skolemize \exists -ASP rules respecting the safety condition as follows: The skolemization of the previous rule results in $\sigma(H) \leftarrow B^+, \text{not } B_1^-, \dots, \text{not } B_k^-$, where $\sigma(H) \leftarrow B^+$ is the skolemization of $H \leftarrow B^+$, as defined for existential rules. The skolemization of an \exists -ASP program is defined by the grounding of Π_F and the skolemization of $\Pi_{\mathcal{R}}$. For example, let $r(X, Z) \leftarrow p(X, Y), \text{not } q(X), \text{not } (r(Y, a), r(a, b))$ be an existential ASP rule. Its skolemization is $r(X, f_Z^R(X)) \leftarrow p(X, Y), \text{not } q(X), \text{not } (r(Y, a), r(a, b))$.

Note that the skolemization of an existential ASP program (without function symbol) is a standard ASP program with function symbols.

Semantics In what follows we consider Π an ASP program obtained from a skolemized existential ASP program. Let \mathcal{C}_{Π} be the set of constants appearing in Π and \mathcal{F}_{Π} be the set of function symbols appearing in Π . The *Herbrand domain* of Π is the minimal set of ground terms \mathcal{H}_{Π} such that $\mathcal{C}_{\Pi} \subseteq \mathcal{H}_{\Pi}$ and, if $f \in \mathcal{F}_{\Pi}$ is a function symbol of arity k and h_1, \dots, h_k are in \mathcal{H}_{Π} , then $f(h_1, \dots, h_k)$ is also in \mathcal{H}_{Π} . If $R = H \leftarrow B^+, \text{not } B_1^-, \dots, \text{not } B_k^-$ is a rule in Π and σ is a substitution from all its variables in to \mathcal{H}_{Π} , then the rule $\sigma(R) = \sigma(H) \leftarrow \sigma(B^+), \text{not } \sigma(B_1^-), \dots, \text{not } \sigma(B_k^-)$ is a *grounding* of R . The grounding of a program Π is the program obtained from all possible groundings of all rules in Π . Note that the Herbrand domain (and thus the grounding) of a finite Π is infinite as soon as Π contains a constant and a predicate symbol of arity ≥ 1 . Let us now consider the grounding Π^G of Π and a (possibly infinite) set of ground atoms E . The *reduct* of Π^G with respect to E , denoted $\Pi_{|E}^G$, is the minimal set that contains all (ground) atoms of Π^G and, for each skolemized \exists -ASP rule $R = H \leftarrow B^+, \text{not } B_1^-, \dots, \text{not } B_k^-$ in Π_G , if there is no B_i^- such that $B_i^- \subseteq E$, then $H \leftarrow B^+$ (called the *positive part* of R) is a skolemized existential rule of $\Pi_{|E}^G$.

Finally, E is an *answer set* (stable model) of Π when $E = Cl(\Pi_{|E}^G)$. We define the answer sets of an existential ASP program as the answer sets of its skolemization. Note that it is not a neutral choice, for a semantic point of view (see the discussion in [4] where using different chases can lead to different semantics and different answer sets).

Computation Given an ASP program Π , most solvers rely upon a 2-step algorithm that first compute the grounding Π^G of Π , then use Π_G to build an answer set E (using for instance a SAT solver). However, the grounding becomes infinite as soon as function symbols (such as the ones obtained from our skolemization) are involved. Some solvers can try to extract from the grounding rules that have no chance to be involved in the second step, but doing that optimally would require to compute that second step, making

the 2-steps separation useless. On the other hand, the ASP solver ASPeRiX [15,14] does not require grounding to compute answer sets (indeed, using homomorphisms during the computation is equivalent to generate the grounding effectively required at that step of a computation). Since our proofs in Section 5 heavily rely upon the soundness and completeness of that algorithm, we explain here its basic version.

In ASPeRiX, given a skolemized existential ASP program Π , a *computation* is an incremental development of a (possibly infinite) binary tree. Each node of this tree contains 3 fields: IN is the set of ground atoms that have been proven in the current branch, OUT is a set of forbidden sets of ground atoms, and MBT (Must Be True) is a set of mandatory disjunctions of sets of ground atoms. Initially, the tree contains a single node, its root, whose IN field contains all ground atoms of Π , and whose fields OUT and MBT are empty. At each step, the computation selects a leaf n of the tree and a rule $R = H \leftarrow B^+, \text{not } B_1^-, \dots, \text{not } B_k^-$ such that there exists a homomorphism σ from B^+ to $\text{IN}(n)$ and (R, σ) has not already been evaluated on n nor on any of its ancestors. Now we say that (R, σ) is evaluated on n and there is 3 possible outcomes. **Blocked case:** If there exists a negative body B_i^- in R such that $\sigma(B_i^-) \subseteq \text{IN}(n)$, meaning that one of the negative bodies appears in $\text{IN}(n)$, then this step produces nothing (but marks this evaluation as done). **Positive case:** If $R = H \leftarrow B^+$ contains no negative body, then we update $\text{IN}(n)$ with the result of the rule application, and do not change OUT nor MBT. Then $\text{IN}(n) = \alpha(\text{IN}(n), R, \sigma) = \text{IN}(n) \cup \sigma(H)$. **Choice case:** otherwise we create two children n_1 and n_2 of n . In n_1 we effectively apply the rule and forbid its negative bodies to appear in the final result, in n_2 we must prove that we have the right not to apply it by finding one of the negative bodies in the final result. Then $\text{IN}(n_1) = \alpha(\text{IN}(n), R, \sigma) = \text{IN}(n) \cup \sigma(H)$, $\text{OUT}(n_1)$ is the set of sets of atoms whose elements are those of $\text{OUT}(n)$ and the k sets of atoms $\sigma(B_i^-)$, for $1 \leq i \leq k$, $\text{MBT}(n_1) = \text{MBT}(n)$ and $\text{IN}(n_2) = \text{IN}(n)$, $\text{OUT}(n_2) = \text{OUT}(n)$, and $\text{MBT}(n_2)$ is the set of disjunctions of sets of atoms whose elements are those of $\text{MBT}(n)$ and the disjunction $\bigvee_{1 \leq i \leq k} \sigma(B_i^-)$.

Consider a (possibly infinite) branch of this tree. Similarly to what was done for derivations, we define the *result* of that branch as the (possibly infinite) union, for all nodes n in that branch, of the $\text{IN}(n)$. When such a branch is finite, its result is $\text{IN}(l)$, where l is the leaf of the branch. A branch is said full when, for every rule R and every homomorphism σ from B^+ to the result of the branch, (R, σ) has been evaluated on some node of the branch. If n is a node of a branch and B is a set of atoms, we say that B satisfies $\text{OUT}(n)$ when, for every set of atoms $O \in \text{OUT}(n)$, $O \not\subseteq B$. In the same way, we say that B satisfies $\text{MBT}(n)$ when, for every disjunction $M_1 \vee \dots \vee M_k \in \text{MBT}(n)$, there exists a M_i such that $M_i \subseteq B$. A branch is said *OUT-valid* (resp. *MBT-valid*) when its result satisfies $\text{OUT}(n)$ (resp. $\text{MBT}(n)$) for every node n in the branch. A branch that is both OUT-valid and MBT-valid is said *valid*.

Theorem 2. *Let Π be a skolemized existential ASP program. Then A is an answer set of Π iff A is the result of a full valid branch in the computation of Π .*

Properties It is first important to note that, when the positive part of rules satisfy the acyclicity conditions presented in [4], then the computation produces a finite tree. In that case, validity of a branch with leaf l admits a simpler characterization: A branch is OUT-valid (resp. MBT-valid) when $\text{IN}(l)$ satisfies $\text{OUT}(l)$ (resp. $\text{MBT}(l)$).

Then we point out the monotonic increase of the field IN: If a node n' is a descendant of a n , then $\text{IN}(n) \subseteq \text{IN}(n')$. It follows that if there is a node n such that $\text{IN}(n)$ does not satisfy $\text{OUT}(n)$, then no branch containing n is OUT-valid, so we can cut the development of the computation tree for node n . Such an optimization is more difficult to achieve using the MBT field, to stop the development of the computation tree for node n , we have to prove that there exists a disjunction $M_1 \vee \dots \vee M_k \in \text{MBT}(n)$ and a set of atoms M_i that will never be contained in the IN field of any descendant of n . Simple arguments achieve that goal in the ASP programs we generate in Section 5.

4 The Notion of Repair

We now recall the definitions of repairs [1,16,10] rephrased within the framework of existential rules. Let $\mathcal{K} = (F, \mathcal{R}, \mathcal{N})$ be a knowledge base where F is a set of ground atoms, \mathcal{R} is a set of existential rules, and \mathcal{N} is a set of negative constraints, *i.e.* a set of rules of form $\perp \leftarrow B$ where B is a set of basic atoms and \perp is the absurd symbol. We say that a set of atoms Y is *consistent* w.r.t. $(\mathcal{R}, \mathcal{N})$ when $(F, \mathcal{R}, \mathcal{N}) \not\models \perp$, *i.e.* when $Cl(Y, \mathcal{R} \cup \mathcal{N})$ does not contain \perp . Our knowledge base is thus consistent when F is consistent w.r.t. $(\mathcal{R}, \mathcal{N})$. Different kind of *repairs* can be considered when the knowledge base is inconsistent. **(Standard) repairs:** A *repair* of \mathcal{K} is an inclusion-maximal subset F' of F that is consistent w.r.t. $(\mathcal{R}, \mathcal{N})$, and we note $F' \in R(\mathcal{K})$. **Closed repairs:** If X is a set of atoms, we call *ground positive closure* of X and note $g^+Cl(X)$ the restriction of $Cl(X, \mathcal{R})$ to basic ground atoms (whose terms are only constants, and not obtained with function symbols). A *closed repair* of \mathcal{K} is a set of basic ground atoms $F'' = g^+Cl(F')$, where F' is a standard repair of \mathcal{K} , and we note $F'' \in CR(\mathcal{K})$. **Repairs of closure:** A *repair of the closure* of \mathcal{K} is a standard repair F' of $(g^+Cl(F, \mathcal{R}), \mathcal{R}, \mathcal{N})$, and we note $F' \in RC(\mathcal{K})$.

Recently a unified framework combining modifiers (way of computing the repairs) and inferences strategies has been proposed for querying ontological knowledge bases represented with existential rules [2]. This framework covers the best known semantics and introduces new ones. The semantics are denoted by $\langle \circ_i, s \rangle$ where \circ_i is a modifier and $s \in \{\forall, \exists, \cap, maj\}$ is an inference strategy. Within this framework \circ_1 computes the set of repairs, \circ_5 computes the closed repairs and \circ_7 computes the repairs of the closure.

5 Computing Repairs with \exists -ASP

In this section we describe the transformation from a knowledge base \mathcal{K} into a generic \exists -ASP program II . Though this program computes “repairs” in the broad sense, two configurable modules (namely selection and display) are used to obtain the intended behaviour. In particular, we show that, given specific rules, this program can compute the repairs, the closed repairs or the repairs of the closure of \mathcal{K} . This transformation relies upon the following steps: 1) \mathcal{K} is put into its skolemized form, 2) the user selects either the select or the display transformation scheme, 3) the transformation builds the program II , using an extended vocabulary, 4) we use an ASP solver to compute the

answer sets of Π , 5) the restriction of those answer sets to the original vocabulary provides the “repairs”.

5.1 Transformation Into \exists -ASP

Our knowledge base is built upon an original vocabulary \mathcal{V} . For every predicate name $p \in \mathcal{V}$, we consider different versions of p that will be used in the extended vocabulary of our \exists -ASP program: p_i for initial predicate, p_p for possible predicate, p_n for forbidden predicate, p_c for chosen predicate, p_s for may be selected predicate, p_v for valid predicate, p_g for ground predicate, and p_d for display predicate. If A is a set of atoms built upon the original vocabulary, we note A_x the set of atoms $p_x(\mathbf{t})$ built upon the extended vocabulary where $p(\mathbf{t})$ is an atom of A . The \exists -ASP program Π is obtained as follows:

Encoding of initial facts : Π contains F_i (every atom of F is considered as an initial fact of the program Π).

Encoding of positive closure : For every predicate name in \mathcal{V} , we have a rule of form $[P_1:] p_p(\mathbf{X}) \leftarrow p_i(\mathbf{X})$, those rules assert that every initial atom is possible; and for every rule $B(\mathbf{X}) \rightarrow H(\mathbf{X}, \mathbf{Y})$ in \mathcal{R}_{sk} , we have a rule of form $[R_1:] H_p(\mathbf{X}, \mathbf{Y}), \text{fact}(Y_1), \dots, \text{fact}(Y_k) \leftarrow B_p(\mathbf{X})$ where the Y_i are the functional terms of the head of the skolemized rule, those rules are used to encode the positive closure $Cl(F, \mathcal{R})$ with possible atoms, and to “mark” functional terms. Finally, for every predicate name $p \in \mathcal{V}$, we have a rule of form $[P_2:] p_g(\mathbf{X}) \leftarrow p_p(\mathbf{X}), \text{not fact}(X_1), \dots, \text{not fact}(X_k)$ asserting that every possible atom using no functional term is ground.

Selection strategy : Those configurable rules provide the user strategy to define which atoms (of form p_s) are selectable, *i.e.* can appear or not in the “repairs”. We provide here two such strategies: **SEL1** says that every initial atom is selectable. For every predicate name $p \in \mathcal{V}$, we have a rule $[S_1:] p_s(\mathbf{X}) \leftarrow p_i(\mathbf{X})$. **SEL2** says that every ground possible atom is selectable. For every predicate name $p \in \mathcal{V}$, we have a rule $[S_2:] p_s(\mathbf{X}) \leftarrow p_g(\mathbf{X})$.

Choice rules : These rules are the core of our program, since they will build all possible subsets of selectable atoms. They say that every atom that is selectable and not forbidden must be chosen. $[P_3:] p_c(\mathbf{X}) \leftarrow p_s(\mathbf{X}), \text{not } p_n(\mathbf{X})$.

Definition of contexts : For every atom $p(\mathbf{t})$, the atom $p_v(\mathbf{t}, c)$ asserts that $p(\mathbf{t})$ is valid in the context c . All chosen atoms are valid in the *base* context. This is encoded, for each predicate name $p \in \mathcal{V}$, by the rule $[P_4:] p_v(\mathbf{X}, \text{base}) \leftarrow p_c(\mathbf{X})$. An atom $p(\mathbf{t})$ that is not chosen will be valid in its own context, encoded by the term $\text{ctx}(p, \mathbf{t})$. This is encoded, for each predicate name $p \in \mathcal{V}$, by the rule $[P_5:] p_v(\mathbf{X}, \text{ctx}(p, \mathbf{X})), \text{context}(\text{ctx}(p, \mathbf{X})) \leftarrow p_s(\mathbf{X}), \text{not } p_c(\mathbf{X})$. Finally, we say that every atom valid in the base context is also valid in any other context. For each predicate name $p \in \mathcal{V}$, we have the rule $[P_6:] p_v(\mathbf{X}, C) \leftarrow p_v(\mathbf{X}, \text{base}), \text{context}(C)$. The base context encodes the chosen atoms. Every other context encodes the adding of one particular unchosen atom to the already chosen ones. Intuitively, to obtain a repair we will have to prove that the base context is consistent and that all other contexts are not, meaning that the base context is maximal.

Context closure : Every atom that can be deduced from those valid in a particular context will also be valid in that context. For every skolemized existential rule of the

form $B(\mathbf{X}) \rightarrow H(\mathbf{X})$, we obtain the rule $[R_2:] H_v(\mathbf{X}, C) \leftarrow B_v(\mathbf{X}, C)$. Then we say that if a constraint is violated in a given context, then that context is *absurd*. For any constraint in \mathcal{N} of the form $p^1(\mathbf{X}_1), \dots, p^k(\mathbf{X}_k) \rightarrow \perp$ we add the rule of form $[C_1:] absurd(C) \leftarrow p_v^1(\mathbf{X}_1, C), \dots, p_v^k(\mathbf{X}_k, C)$.

Retropropagation of absurd contexts : Finally, we say that if the base context is absurd, then every atom valid in that context is forbidden. For every predicate $p \in \mathcal{V}$, we have the rule $[C_2:] p_n(\mathbf{X}) \leftarrow p_c(\mathbf{X}), absurd(base)$. For other absurd contexts, only selectable unchosen atoms of that specific context are forbidden. For every predicate $p \in \mathcal{V}$, we have the rule $[C_3:] p_n(\mathbf{X}) \leftarrow not p_c(\mathbf{X}), p_s(\mathbf{X}), p_v(\mathbf{X}, C), context(C), absurd(C)$.

Visualization strategy : Those configurable rules provide the user strategy to define which atoms (of form p_d) are displayable, *i.e.* can appear or not in the visualization of the “repairs”. Whatever the strategy chosen, only displayable atoms that are valid in the base context will be displayed (*i.e.* added using the original vocabulary). This is encoded, for each predicate name $p \in \mathcal{V}$, by the rule $[D:] p(\mathbf{X}) \leftarrow p_d(\mathbf{X}), p_v(\mathbf{X}, base)$. We provide here two such strategies: **DISP1** says that every initial atom is displayable. For every predicate name $p \in \mathcal{V}$, we have a rule $[V_1:] p_d(\mathbf{X}) \leftarrow p_i(\mathbf{X})$. **DISP2** says that every ground possible atom is displayable. For every predicate name $p \in \mathcal{V}$, we have a rule $[V_2:] p_d(\mathbf{X}) \leftarrow p_g(\mathbf{X})$.

Example 1. Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base such that $F = \{p(a), q(a)\}$, $\mathcal{R}_{sk} = \{p(X) \rightarrow r(X, f(X)), q(X) \rightarrow s(X), r(X, Y) \rightarrow t(X)\}$ and $\mathcal{N} = \{r(X, Y), q(X) \rightarrow \perp\}$. The original vocabulary of \mathcal{K} contains the predicate names $\{p, q, r, t\}$.

The **initial facts** are $p_i(a)$. and $q_i(a)$.

The rules encoding the **positive closure** are those of form P_1 for initialization (we restricted those to the predicates appearing in initial form): $p_p(X) \leftarrow p_i(X)$. and $q_p(X) \leftarrow q_i(X)$., those of form R_1 for propagation: $r_p(X, f(X)), fct(f(X)) \leftarrow p_p(X)$. $s_p(X) \leftarrow q_p(X)$. $t_p(X) \leftarrow r_p(X, Y)$. and those of form P_2 to detect ground atoms: $p_g(X) \leftarrow p_p(X), not fct(X)$. $q_g(X) \leftarrow q_p(X), not fct(X)$. $r_g(X, Y) \leftarrow r_p(X, Y), not fct(X), not fct(Y)$. $s_g(X) \leftarrow s_p(X), not fct(X)$. $t_g(X) \leftarrow t_p(X), not fct(X)$.

Two **selection strategies** are possible. With **SEL1** we have: $p_s(X) \leftarrow p_i(X)$. and $q_s(X) \leftarrow q_i(X)$. With **SEL2** we have: $p_s(X) \leftarrow p_g(X)$. $q_s(X) \leftarrow q_g(X)$. $r_s(X, Y) \leftarrow r_g(X, Y)$. $s_s(X) \leftarrow s_g(X)$. and $t_s(X) \leftarrow t_g(X)$.

The **choice rules** are: $p_c(X) \leftarrow p_s(X), not p_n(X)$. $q_c(X) \leftarrow q_s(X), not q_n(X)$. $r_c(X, Y) \leftarrow r_s(X, Y), not r_n(X, Y)$. $s_c(X) \leftarrow s_s(X), not s_n(X)$. $t_c(X) \leftarrow t_s(X), not t_n(X)$.

For the **definition of contexts**, we have the rules of form P_4 defining the base context: $p_v(X, base) \leftarrow p_c(X)$. $q_v(X, base) \leftarrow q_c(X)$. $r_v(X, Y, base) \leftarrow r_c(X, Y)$. $s_v(X, base) \leftarrow s_c(X)$. $t_v(X, base) \leftarrow t_c(X)$. the rules of form P_5 defining other contexts: $p_v(X, ctx(p, X)), context(ctx(p, X)) \leftarrow p_s(X), not p_c(X)$. $q_v(X, ctx(q, X)), context(ctx(q, X)) \leftarrow q_s(X), not q_c(X)$. $r_v(X, Y, ctx(r, X, Y)), context(ctx(r, X, Y)) \leftarrow r_s(X, Y), not r_c(X, Y)$. $s_v(X, ctx(s, X)), context(ctx(s, X)) \leftarrow s_s(X), not s_c(X)$. $t_v(X, ctx(t, X)), context(ctx(t, X)) \leftarrow t_s(X), not t_c(X)$. and the rules of form P_6 encoding inheritance of base context: $p_v(X, C) \leftarrow p_v(X, base), context(C)$. $q_v(X, C) \leftarrow q_v(X, base), context(C)$. $r_v(X, Y, C) \leftarrow r_v(X, Y, base), context(C)$. $s_v(X, C)$

$\leftarrow s_v(X, base), context(C). t_v(X, C) \leftarrow t_v(X, base), context(C).$

The **context closure** will be computed with the rules of form $R_2: r_v(X, f(X), C) \leftarrow p_v(X, C). s_v(X, C) \leftarrow q_v(X, C). t_v(X, C) \leftarrow r_v(X, Y, C).$ and inconsistencies will be detected by the rule of form $C_1: absurd(C) \leftarrow r_v(X, Y, C), q_v(X, C).$

Retropropagation of absurd contexts is handled by rules of form $C_2: p_n(X) \leftarrow p_c(X), absurd(base). q_n(X) \leftarrow q_c(X), absurd(base). r_n(X, Y) \leftarrow r_c(X, Y), absurd(base). s_n(X) \leftarrow s_c(X), absurd(base). t_n(X) \leftarrow t_c(X), absurd(base).$ and $C_3: p_n(X) \leftarrow not p_c(X), p_s(X), p_v(X, C), context(C), absurd(C). q_n(X) \leftarrow not q_c(X), q_s(X), q_v(X, C), context(C), absurd(C). r_n(X, Y) \leftarrow not r_c(X, Y), r_s(X, Y), r_v(X, Y, C), context(C), absurd(C). s_n(X) \leftarrow not s_c(X), s_s(X), s_v(X, C), context(C), absurd(C). t_n(X) \leftarrow not t_c(X), t_s(X), t_v(X, C), context(C), absurd(C).$ Finally, **display rules** contain the rules of form $D: p(X) \leftarrow p_d(X), p_v(X, base). q(X) \leftarrow q_d(X), q_v(X, base). r(X, Y) \leftarrow r_d(X, Y), r_v(X, Y, base). s(X) \leftarrow s_d(X), s_v(X, base). t(X) \leftarrow t_d(X), t_v(X, base).$ And the choice of strategy **DISP1** with rules: $p_d(X) \leftarrow p_i(X). q_d(X) \leftarrow q_i(X). r_d(X, Y) \leftarrow r_i(X, Y). s_d(X) \leftarrow s_i(X). t_d(X) \leftarrow t_i(X).$ or of strategy **DISP2** with rules: $p_d(X) \leftarrow p_g(X). q_d(X) \leftarrow p_g(X). r_d(X, Y) \leftarrow r_g(X, Y). s_d(X) \leftarrow s_g(X). t_d(X) \leftarrow t_g(X).$

It is important to note that when the skolem chase halts for the original existential rules KB (such fragments have been studied for instance in [4]) then the Skolem chase also halts on the positive part of the generated ASP program, and thus (see properties in Section 3) the ASPeRiX computation generates all answer sets in finite time.

5.2 General Form of the Computation Tree of Π

Let us now examine what is happening during a computation of such a program Π . We first point out that we can evaluate rules in a particular order: 1) the positive closure rules of form P_1 , 2) those of form R_1 , 3) those of form P_2 , 4) the selection rules, 5) the choice rules P_3 , 6) the definitions of contexts of form P_4 , 7) those of form P_5 , 8) those of form P_6 , 9) the context closure of form R_2 , 10) and those of form C_1 , 11) the retropropagation rules C_2 and 12) C_3 , and 13) the visualisation rules. Indeed, we can check that, if $i < j$ are two of those steps, no rule evaluated at step j can trigger a new application of a rule that was evaluated at step i . This will not always be the case with any selection rules provided by the user, but this property is satisfied by the strategies **SEL1** and **SEL2** presented here. Among all equivalent computation trees, we will thus consider those that respect that particular order: The *natural* computations of Π .

Proposition 1. *Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base, and let Π be the \exists -ASP program obtained from the above encoding. At the end of Step 3 the natural computation tree corresponding to Π only has one finite branch that could lead to a full valid branch.*

Proof. The computation of Π is a binary tree. Initially the root is s.t $IN(root) = \mathcal{F}_i$, $OUT(root) = \emptyset$, $MBT(root) = \emptyset$. After $|\mathcal{F}_i|$ applications of the rule P_1 , $IN(root) = \mathcal{F}_i \cup \mathcal{F}_p$, since the rule P_1 is positive (the negative body of P_1 is empty) $OUT(root)$ and $MBT(root)$ are unchanged. (In the following in case of positive rule we do not specify that the fields OUT and MBT do not change.) After a possible infinite number of applications of the rule R_1 , $IN(root) = \mathcal{F}_i \cup (Cl(\mathcal{F}, \mathcal{R}))_p \cup \{fct(t), t \notin \text{basic terms of}$

$Cl(\mathcal{F})$. We develop the computation tree using the rule P_2 , starting from the root, for each node n we look for a homomorphism σ in $IN(root)$ s.t $\sigma(X_i)=t_i$ where t_i is a grounded term. Two cases hold:

- case 1: $\exists t_i$ such that $fact(t_i) \in IN(root)$. This is the blocked case of the computation tree given in Section 3. The node is not changed.
- case 2: $\nexists t_i$ such that $fact(t_i) \in IN(root)$. This is the choice case in the computation tree given in Section 3. The node n has two children n_1 and n_2 such that $IN(n_1) = IN(n) \cup \{p_g(t_1, \dots, t_k)\}$, $OUT(n_1) = OUT(n) \cup \{\{fact(t_1)\}, \dots, \{fact(t_k)\}\}$, $MBT(n_1) = MBT(n)$ and $IN(n_2) = IN(n)$, $OUT(n_2) = OUT(n)$, $MBT(n_2) = MBT(n) \cup \{fact(t_1) \vee \dots \vee fact(t_k)\}$.

Note that we get all $fact(t_i)$ that could be generated and there will be no other way to obtain others. According to the properties in Section 3 none of the $fact(t_i)$ in $MBT(n_2)$ can be proved therefore this branch cannot lead to a valid branch. At the end of Step 3, the computation tree only has one branch that could lead to a valid branch and therefore to an answer set. Since there is a finite number of atoms without function symbol, this only branch is finite and l denotes its leaf and $IN(l) = \mathcal{F}_i \cup (Cl(\mathcal{F}, \mathcal{R}))_p \cup \{fact(t) \mid t \text{ is a functional term of } Cl(\mathcal{F}, \mathcal{R})\} \cup (\{a \in Cl(\mathcal{F}, \mathcal{R}) \mid a \text{ is a basic atom}\})_g$, $OUT(l) = \{\{fact(t)\} \mid t \text{ is a functional term of } Cl(\mathcal{F}, \mathcal{R})\}$ and $MBT(l) = MBT(n)$. As no further development of the computation tree can add any atom with predicate name $fact(t)$, the result of any branch having the node l as ancestor will satisfy $OUT(l)$. Thus, in the following, we will ignore $OUT(l)$.

Example 2. (Example 1, continued) At the end of Step 3 the computation tree has only one branch and l denotes its leaf. We have $IN(l) = \{p_i(a), q_i(a), p_p(a), q_p(a), r_p(a), f(a), fact(f(a)), s_p(a), t_p(a), p_g(a), q_g(a), s_g(a), t_g(a)\}$, $OUT(l) = \{\{fact(a)\}\}$ and $MBT(l) = \{fact(f(a))\}$. Note that this branch may lead to a full valid branch since $IN(l)$ satisfies $OUT(l)$ and $IN(l)$ satisfies $MBT(l)$.

Proposition 2. *Let $\mathcal{K}=(\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base, and let Π be the \exists -ASP program obtained from the above encoding. Let X be the finite set of selectable atoms obtained after Step 4. At the end of Step 5 the natural computation tree corresponding to Π has $2^{|X|}$ finite branches (each one determined by the subset Y of the chosen atoms in X).*

Proof. As shown in Proposition 1 the computation tree corresponding to Π obtained at the end of Step 3 only has one finite branch and l denotes its leaf. We start from l where $IN(l)$, $OUT(l)$ and $MBT(l)$ are given at the end of the proof of Proposition 1. Step 4 proposes two strategies for selecting the predicates, in order to handle both cases, we consider the set of atoms X provided by the selection rules and the field IN is updated with X . Thanks to the proposed selection rules X is always finite. The application of the rules P_3 leads to the development of $2^{|X|}$ sub-branches from l , each one encoding a subset $Y \subseteq X$. The branch associated with Y has a leaf denoted by l_{Y_1} such that $IN(l_{Y_1}) = IN(l) \cup Y$ denoted by IN_{Y_1} , $OUT(l_{Y_1}) = OUT(l) \cup \{\{p_n(\mathbf{t}) \mid p_s(\mathbf{t}) \in Y\}\}$ and $MBT(l_{Y_1}) = MBT(l) \cup \{(X \setminus Y)_n\}$.

Proposition 3. *Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base, and let Π be the \exists -ASP program obtained from the above encoding. Let l_{Y_1} be the leaf of a branch obtained after Step 5 of the natural computation tree. Then l_{Y_1} can lead to at most one valid full branch, which is finite.*

Proof. We now consider the development of the computation tree from l_{Y_1} . The application of the rules P_4 introduces the *base* context and since they are positive only the field IN is updated, thus $IN(l_{Y_1}) = INY1 \cup \{p_v(\mathbf{t}, base) \mid p(\mathbf{t}) \in Y\}$. The rules P_5 introduce the contexts different from the *base* context. These are choice rules however like in the case of rules P_2 no other application of rules can generate chosen predicates ($p_c(\mathbf{t})$) therefore there is only one branch that can eventually lead to a valid branch and l_{Y_2} denotes its leaf. Note that this branch is finite because X is finite. Thus $IN(l_{Y_2}) = IN(l_{Y_1}) \cup \{p_v(\mathbf{t}, ctx(p, \mathbf{t})) \mid p(\mathbf{t}) \in X \setminus Y\} \cup \{context(ctx(p, \mathbf{t})), \mid p(\mathbf{t}) \in X \setminus Y\}$, denoted by $INY2$, the fields OUT and MBT are unchanged. The application of the rules P_6 updates the field IN , thus $IN(l_{Y_2}) = INY2 \cup \{p_v(\mathbf{t}, c) \mid p(\mathbf{t}) \in Y\}$, denoted by $NY3$, where c is a constant different from *base*. The application of rules R_2 updates the field IN , thus $IN(l_{Y_2}) = INY3 \cup \{p_v(\mathbf{t}, base) \mid p(\mathbf{t}) \in Cl(Y, \mathcal{R})\} \cup \{p_v(\mathbf{t}, c) \mid c = ctx(q, \mathbf{u}), c \neq base \text{ and } p(\mathbf{t}) \in Cl(Y \cup \{q(\mathbf{u})\}, \mathcal{R})\}$, denoted by $INY4$. The application of the rules C_1 updates the field IN , thus $IN(l_{Y_2}) = INY4 \cup \{absurd(base) \mid Cl(Y, \mathcal{R}) \text{ violates a constraint}\} \cup \{absurd(c) \mid c = ctx(q, \mathbf{u}), c \neq base \text{ and } Cl(Y \cup \{q(\mathbf{u})\}, \mathcal{R}) \text{ violates a constraint}\}$, denoted by $INY5$. The application of the rules C_2 updates the field IN , thus $IN(l_{Y_2}) = INY5 \cup Y_n$ if $Cl(Y, \mathcal{R})$ violates a constraint or $IN(l_{Y_2}) = INY5$ otherwise. The rules C_3 introduce the forbidden predicates These are choice rules however like in the case of rules P_4 no other application of rules can generate chosen predicates ($p_c(\mathbf{t})$) therefore there is only one branch that can eventually lead to a valid branch. l_{Y_3} denotes its leaf. Note that this branch is finite because X is finite. Thus $IN(l_{Y_3}) = IN(l_{Y_2}) \cup \{p_n(\mathbf{t}) \mid Cl(Y \cup p_c(\mathbf{t}), \mathcal{R}) \text{ violates a constraint}\}$ denoted by $INY6$, the fields OUT and MBT are unchanged. Step 13 proposes two strategies for visualizing the predicates, with the strategy **DISP1** the field IN is updated such that $IN(l_{Y_3}) = INY6 \cup \{p_d(\mathbf{t}) \mid p_i(\mathbf{t}) \in \mathcal{F}_i\}$, while with the strategy **DISP2** the field IN is updated such that $IN(l_{Y_3}) = INY6 \cup \{p_d(\mathbf{t}) \mid p_g(\mathbf{t}) \in (Cl(\mathcal{F}, \mathcal{R}))_p\}$. Finally the display rule D updates the field IN , thus $IN(l_{Y_3}) = INY7 \cup \{p(\mathbf{t})\}$ where p is valid in the *base* context and $p_d(\mathbf{t})$ has been selected by a visualization strategy. At the end of Step 13, the branch associated with Y is full. The computation is finite even if its nodes can require an infinite derivation.

5.3 Computation Tree of Π and Repairs

As a preliminary remark, and since all the branches of the computation tree are finite, let us point out that we can thus use the characterization of the validity given in the properties of Section 3 using the leaves of that tree. The branch associated with Y is *OUT – valid* if and only if $IN(l_{Y_3})$ satisfies $OUT(l_{Y_3})$. Moreover, the branch associated with Y is *MBT – valid* if and only if $IN(l_{Y_3})$ satisfies $MBT(l_{Y_3})$.

Theorem 3. *Let $\mathcal{K}=(\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base. Let Π be the \exists -ASP program obtained from \mathcal{K} according to the above encoding. Let Y be a subset of the set of selectable*

atoms X . The full branch of the computation tree corresponding to Π , associated with Y is valid if and only if Y is a maximal subset of X such that $Cl(Y, \mathcal{R} \cup \mathcal{N}) \not\models \perp$.

Proof. By hypothesis $Y \subseteq X$, thus by Proposition 3 the computation tree provides a full branch associated with Y and l denotes its leaf. We prove the first the direction by contraposition. If $Cl(Y, \mathcal{R} \cup \mathcal{N}) \models \perp$ then $\exists N \in \mathcal{N}$ such that $Cl(Y, \mathcal{R}) \models N$ thus $absurd(base) \in IN(l)$, thus $\forall p(\mathbf{t}) \in Y$ we have $p_n(\mathbf{t}) \in IN(l)$ and $p_n(\mathbf{t}) \in OUT(l)$ therefore the branch associated with Y is not *OUT-valid*. Suppose now that $Cl(Y, \mathcal{R} \cup \mathcal{N}) \not\models \perp$ but there exists $p(\mathbf{t}) \in X \setminus Y$ s.t $Cl(Y \cup \{p(\mathbf{t})\}, \mathcal{R} \cup \mathcal{N}) \models \perp$ thus $p_v(\mathbf{t}, ctx(p, \mathbf{t})) \in IN(l)$ and we cannot obtain $absurd(ctx(p, \mathbf{t}))$. However $p_n(\mathbf{t})$ could only be obtained from $absurd(ctx(p, \mathbf{t}))$, $p_n(\mathbf{t}) \notin IN(l)$ but since $p(\mathbf{t}) \in X \setminus Y$, $p_n(\mathbf{t}) \in MBT(l)$ therefore the branch is not *MBT-valid*.

We now prove the other direction. Let Y be a maximal subset of X such that $Cl(Y, \mathcal{R} \cup \mathcal{N}) \not\models \perp$. Thus $absurd(base) \notin IN(l)$ and $\forall p(\mathbf{t}) \in Y$, $p_n(\mathbf{t}) \notin IN(l)$. Since $OUT(l) = \{p_n(\mathbf{t}) \mid p(\mathbf{t}) \in Y\}$ then the branch associated with Y is *OUT-valid*. Y is maximal w. r. t. set inclusion thus $\forall q(\mathbf{u}) \in X \setminus Y$ we have $Cl(Y \cup \{q(\mathbf{u})\}, \mathcal{R} \cup \mathcal{N}) \models \perp$, thus $absurd(ctx(q, \mathbf{u})) \in IN(l)$, thus $q_n(\mathbf{u}) \in IN(l)$ and since $q(\mathbf{u}) \in X \setminus Y$ then $q_n(\mathbf{u}) \in MBT(l)$ therefore the branch associated with Y is *MBT-valid*.

We did not discuss yet the effects of the selection and visualization strategies on the results of our program. If we select the atoms with Strategy **SEL1** then X is exactly the set F . If we select the atoms with Strategy **SEL2** then X is exactly the ground closure of F . According to Theorem 3, using Strategy **SEL1** the result of the branch associated with Y is an answer if and only if Y is maximal consistent subset of F while using Strategy **SEL2** the result of the branch associated with Y is an answer if and only if Y is maximal consistent subset of the ground closure of F . When displaying atoms with Strategy **DISP1** the restriction of the answer set associated with a branch Y to the predicates of the original vocabulary is exactly $Cl(Y, \mathcal{R}) \cap F$ while displaying the atoms with Strategy **DISP2** the restriction of the answer set associated with a branch Y to the predicates of the original vocabulary is exactly $Cl(Y, \mathcal{R})$. Let Π be an \exists -ASP program obtained from the above encoding. Let AS be an answer set of Π , $\rho(AS)$ denotes the restriction of AS to the original vocabulary \mathcal{V} and $\rho(\Pi) = \{\rho(AS) \mid AS \in AS(\Pi)\}$.

Corollary 1. Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be knowledge base. Let Π_1 be the \exists -ASP program obtained from the above encoding using strategies **SEL1** and **DISP1**. Then $\rho(\Pi_1)$ is the set of repairs of \mathcal{K} . Let Π_5 be the \exists -ASP program obtained from the above encoding using strategies **SEL1** and **DISP2**. Then $\rho(\Pi_5)$ is the set of closed repairs of \mathcal{K} . Let Π_7 be the \exists -ASP program obtained from the above encoding using strategies **SEL2** and **DISP2**. Then $\rho(\Pi_7)$ is the set of repairs of the closure of \mathcal{K} .

Example 3. The selection strategy **SEL1** allows one to select the predicates in F and provides the set $X = \{p_s(a), q_s(a)\}$. The computation tree develops 4 branches, each one encoding a subset of Y of X . Only two of them are full valid branches. The selection strategy **SEL2** allows one to select the predicates in the grounded closure of F and provides the set $X = \{p_s(a), q_s(a), s_s(a), t_s(a)\}$. The computation tree develops 16 branches, each one encoding a subset of Y of X . Only two of them are full valid branches. The visualization strategy **DISP1** allows one to display valid predicates

within the *base* context which belong to F while the visualization strategy **DISP2** allows one to display valid predicates within the *base* context which belong to grounded closure of F . Using strategies **SEL1** and **DISP1** we obtain an \exists -ASP program denoted by Π_1 such that the answer sets restricted to the original vocabulary are $\{p(a)\}$ and $\{q(a)\}$. Note that they correspond to the repairs of \mathcal{K} . Using strategies **SEL1** and **DISP2** we obtain an \exists -ASP program denoted by Π_5 s.t the answer sets restricted to the original vocabulary are $\{p(a), t(a)\}$ and $\{q(a), s(a)\}$. Note that they correspond to the closed repairs of \mathcal{K} . Using strategies **SEL2** and **DISP1** we obtain an \exists -ASP program denoted by Π_7 s.t the answer sets restricted to the original vocabulary are $\{p(a), s(a), t(a)\}$ and $\{q(a), s(a), t(a)\}$. Note that they correspond to the repairs of the closure of \mathcal{K} .

5.4 Other Strategies

We have presented here a generic encoding of a knowledge base \mathcal{K} into an ASP program that computes different kind of repairs of \mathcal{K} , according to the different selection rules and display rules we have chosen in that encoding. This generic ASP program could take into account other possible select/display rules to achieve different outcome. For instance, let us consider the following set of rules. **Selection rules:** The user defines all “optional” atoms with rules of form $p_s(\mathbf{X}) \leftarrow p_i(\mathbf{X})$., where all atoms of F with predicate name p are optional and $p_s(\mathbf{a}) \leftarrow p_i(\mathbf{a})$., where the atom $p(\mathbf{a})$ of F is optional and then asserts that every atom of F that is not optional is mandatory. For every predicate name p , there is a rule of form $p_v(\mathbf{X}, base) \leftarrow p_i(\mathbf{X}), not p_s(\mathbf{X})$. **Display rules:** The user can use rules similar to the selection rules to display only optional atoms of F . With such a set of select/display rules, the program Π will admit an answer set only when the subset M of mandatory atoms of F (i.e. those that are not declared optional) is consistent w.r.t. $(\mathcal{R}, \mathcal{N})$, and in that case, if AS is an answer set of Π , $\rho(AS)$ will be an inclusion-maximal subset F' of F such that $M \cup F'$ is consistent w.r.t. $(\mathcal{R}, \mathcal{N})$.

6 Conclusion

This paper presented a generic encoding in \exists -ASP of repair-based techniques for inconsistent knowledge bases expressed within the formalism of existential rules. We focused on three kinds of repairs that allow for computing query answering with the following semantics proposed in [2]: $\langle \circ_1, \forall \rangle$ (corresponds to AR-semantics [16]), $\langle \circ_1, \cap \rangle$ (corresponds to IAR-semantics [16]), $\langle \circ_7, \forall \rangle$ (close to CAR-semantics [16]), $\langle \circ_7, \cap \rangle$ (close to ICAR-semantics [16]) and $\langle \circ_5, \cap \rangle$ (corresponds to ICR-semantics [10]). Indeed these semantics can be rephrased in our framework as follows. Let \mathcal{K} be a knowledge base and let q and q_v be first order formulas, where q_v is obtained from q by replacing each predicate $p(\mathbf{t})$ occurring in q by $p_v(\mathbf{t}, base)$ we have: 1) $\mathcal{K} \models_{\langle \circ_1, \forall \rangle} q$ iff $\forall AS \in AS(\Pi_1), q_v \in AS$. 2) $\mathcal{K} \models_{\langle \circ_1, \cap \rangle} q$ iff $q_v \in \cap_{AS_i \in AS(\Pi_1)} AS_i$. 3) $\mathcal{K} \models_{\langle \circ_7, \forall \rangle} q$ iff $\forall AS \in AS(\Pi_7), q_v \in AS$. 4) $\mathcal{K} \models_{\langle \circ_7, \cap \rangle} q$ iff $q_v \in \cap_{AS_i \in AS(\Pi_7)} AS_i$. 5) $\mathcal{K} \models_{\langle \circ_5, \cap \rangle} q$ iff $q_v \in \cap_{AS_i \in AS(\Pi_5)} AS_i$.

A future work will be dedicated to the implementation and experimentation of the proposed encoding with *ASPeRiX* [14]. Another interesting issue is the extension of this encoding to the modifiers proposed within the unified framework for inconsistency-tolerant query answering stemming from the selection modifier based on cardinality.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proc. of SIGACT-SIGMOD-SIGART. pp. 68–79 (1999)
2. Baget, J.F., Benferhat, S., Bouraoui, Z., Croitoru, M., Mugnier, M.L., Papini, O., Rocher, S., Tabia, K.: A general modifier-based framework for inconsistency-tolerant query answering. In: Proc. of KR'16 (2016)
3. Baget, J., Garreau, F., Mugnier, M., Rocher, S.: Extending acyclicity notions for existential rules. In: Proc of ECAI'14. pp. 39–44 (2014)
4. Baget, J., Garreau, F., Mugnier, M., Rocher, S.: Revisiting chase termination for existential rules and their extension to nonmonotonic negation. In: Proc. of NMR'14 (2014)
5. Baget, J., Leclère, M., Mugnier, M., Salvat, E.: On rules with existential variables: Walking the decidability line. *Artif. Intell.* 175(9-10), 1620–1654 (2011)
6. Baral, C.: Knowledge Representation Reasoning and Declarative Problem Solving. Cambridge University Press (2008)
7. Benferhat, S., Bouraoui, Z., Tabia, K.: How to select one preferred assertional-based repair from inconsistent and prioritized dl-lite knowledge bases? In: Proc.of IJCAI'15. pp. 1450–1456 (2015)
8. Benferhat, S., Dubois, D., Prade, H.: Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case. *Studia Logica* 58(1), 17–45 (1997)
9. Bertossi, L.E.: Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management, Morgan & Claypool (2011)
10. Bienvenu, M.: On the complexity of consistent query answering in the presence of simple ontologies. In: Proc. of AAAI'12 (2012)
11. Bienvenu, M., Bourgaux, C., Goasdoué, F.: Querying inconsistent description logic knowledge bases under preferred repair semantics. In: Proc. of AAAI'16 (2016)
12. Bienvenu, M., Rosati, R.: Tractable approximations of consistent query answering for robust ontology-based data access. In: Proc. of IJCAI'2013 (2013)
13. Garreau, F., Garcia, L., Lefèvre, C., Stéphane, I.: \exists -asp. In: Proc. of JOWO'15 (2015)
14. Lefèvre, C., Béatrix, C., Stéphane, I., Garcia, L.: Asperix, a first order forward chaining approach for answer set computing. CoRR (to appear in TPLP) abs/1503.07717 (2015)
15. Lefèvre, C., Nicolas, P.: A first order forward chaining approach for answer set computing. In: Proc. of LPNMR 2009. pp. 196–208 (2009)
16. Lembo, D., M.Lenzerini, Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.* 33, 3–29 (2015)
17. Lukasiewicz, T., Martinez, M.V., Pieris, A., Simari, G.I.: From classical to consistent query answering under existential rules. In: Proc. of AAAI'15. pp. 1546–1552 (2015)
18. Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Inconsistency handling in datalog \pm ontologies. In: Proc. of ECAI'12. pp. 558–563 (2012)
19. Rosati, R.: On the complexity of dealing with inconsistency in description logic ontologies. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence IJCAI'11. pp. 1057–1062 (2011)
20. Wan, H., Zhang, H., Xiao, P., Huang, H., Zhang, Y.: Query answering with inconsistent existential rules under stable model semantics. In: Proc. of AAAI'16 (2016)