Existential quantifiers in the rule body

Pedro Cabalar*

Department of Computer Science,
Corunna University (Corunna, Spain),
cabalar@udc.es

Abstract. In this paper we consider a simple syntactic extension of Answer Set Programming (ASP) for dealing with (nested) existential quantifiers and double negation in the rule bodies, in a close way to the recent proposal RASPL-1. The semantics for this extension just resorts to Equilibrium Logic (or, equivalently, to the General Theory of Stable Models), which provides a logic-programming interpretation for any arbitrary theory in the syntax of Predicate Calculus. We present a translation of this syntactic class into standard logic programs with variables (either disjunctive or normal, depending on the input rule heads), as those allowed by current ASP solvers. The translation relies on the introduction of auxiliary predicates and the main result shows that it preserves strong equivalence modulo the original signature.

1 Introduction

One of the traditional limitations of Answer Set Programming (ASP) in the past has been the need of resorting to a ground instantitation of program rules. Starting from the original definition of Stable Models [1] in terms of a propositional language, ASP solvers were designed following a two step process: first, removing variables in favour of all their ground instances; and second, computing the stable models of the resulting ground program. Variables were somehow an "external" element that was not directly treated in the semantics. It is not surprising, in this way, that quantification was not paid too much attention in the past although, paradoxically, most practical applications of ASP deal in one way or another with some limited use of quantified variables, using auxiliary predicates to capture the intended meaning.

This general picture has experienced a drastical change in the last years thanks to the introduction of Quantified Equilibrium Logic [2] (QEL) or the equivalent definition of stable models for first-order formulas proposed in [3]. These approaches provide a logic-programming interpretation for any arbitrary first-order theory, so that syntactic restrictions do not play a role in the semantic definition any more. Some recent results have been obtained in applying this semantics to programs with variables, without resorting to grounding. For instance, [4] treats the problem of strong equivalence, whereas in [5] QEL is used

* This research was partially supported by Spanish MEC project TIN-2006-15455-C03-02 and Xunta de Galicia project INCITE08-PXIB105159PR.

to analyse rule redundancy and the completeness of rule subsumption under a given substitution. On the other hand, much work remains to be done yet in exploring the intuition, under a logic-programming perspective, of the QEL interpretation of formulas with arbitrary syntax or belonging to new syntactic classes. Several works have followed this direction: we can mention [6], that has studied the extension of the concept of *safety* for arbitrary theories; [7], which considers an extension for dealing with partial functions; or [8], that proposes a logic-programming language RASPL-1 for counting and choice that can be translated into first-order expressions under QEL by introducing existential quantifiers and double negations in the rule bodies.

In this paper we analyse an extension of logic programs with variables where, similarly to first-order theories resulting from the RASPL-1 translation, we introduce existential quantifiers and double negations in the rule bodies, further allowing a way of nesting these new constructs (something not considered in [8]). We provide some intutitions of the utility of this extension and explain how these features are already used in the current ASP programming style by a suitable introduction of auxiliary predicates. In fact, we propose an automated translation that relies on this technique of auxiliary predicates and reduces the proposed extension to regular logic programs with variables as those accepted by current ASP grounding tools. This translation is shown to be strongly equivalent (modulo the original language without the auxiliary predicates). Apart from providing a more readable and compact representation, the advantage of dealing with the extended syntax is avoiding a potential source of errors in the introduction of auxiliary predicates, not only due to a possible programmer's mistake in the formulation, but especially because auxiliary predicates must be guaranteed to be hidden and limited to their original use.

The rest of the paper is organised as follows. In the next section we introduce some motivating examples and explain the paper goals. In Section 3 we provide an overview of Quantified Equilibrium Logic to proceed in the next section with the introduction of the syntactic subclass we study in this paper. Section 5 presents the translation of this class into regular logic programs, proving its correctnes. Section 6 discusses some related work and finally, Section 7 concludes the paper.

2 Motivation

Example 1. Given the extent of predicates $person(X)$, $parent(X, Y)$ (X is a parent of Y) and $married(X, Y)$ which is a symmetric relation, suppose we want to represent that a person is happy, $happy(X)$, when all his/her offsprings are married. □

A typical piece of program representing this problem in ASP would probably look like:

$$\begin{aligned} \mathit{has\_spouse}(Y) &\leftarrow \mathit{married}(Y, Z) \\ \mathit{has\_single\_offs}(X) &\leftarrow \mathit{parent}(X, Y), \mathit{not}\ \mathit{has\_spouse}(Y) \\ \mathit{happy}(X) &\leftarrow \mathit{person}(X), \mathit{not}\ \mathit{has\_single\_offs}(X) \end{aligned}$$

Notice how predicates *has_spouse* and *has_single_offs* are *not* in the problem enunciate. Furthermore, their name suggest that we are capturing an existential quantifier: note how in the first two rules, we have a free variable in the body that does not occur in the head. In other words, a more compact way of representing this program could just be:

$$\mathit{happy}(X) \leftarrow \mathit{person}(X), \mathit{not}\ \exists Y(\mathit{parent}(X, Y), \mathit{not}\ \exists Z\ \mathit{married}(Y, Z)) \quad (1)$$

We will show that, in fact, both representations are strongly equivalent under QEL if we restrict the use of the auxiliary predicates *has_spouse* and *has_single_offs* to the above mentioned rules. Notice, however, the importance of this second representation. We, not only, get a more readable formula and avoid auxiliary predicates not included in the original problem: we also avoid a possible mistake in the use of these predicates in another part or module of the program, something that could radically change their intended meaning for the example.

As another typical example of an implicit existential quantifier, consider the frequent formalisation of the inertia default in ASP:

$$\mathit{holds}(F, V, \mathit{do}(A, S)) \leftarrow \mathit{holds}(F, V, S), \mathit{not}\ \mathit{ab}(F, V, A, S) \quad (2)$$

$$\mathit{ab}(F, V, A, S) \leftarrow \mathit{holds}(F, W, \mathit{do}(A, S)), W \neq V \quad (3)$$

where the complete rule bodies would also include the atoms $\mathit{action}(A)$, $\mathit{situation}(S)$, $\mathit{fluent}(F)$, $\mathit{range}(F, V)$ and $\mathit{range}(F, W)$ to specify the sorts of each variable[1]. Again, predicate *ab* is introduced to capture the meaning: "there exists a value for F other than V." In other words, the formula could have been written instead as:

$$\mathit{holds}(F, V, \mathit{do}(A, S)) \leftarrow \mathit{holds}(F, V, S), \mathit{not}\ \exists W(\mathit{holds}(F, W, \mathit{do}(A, S)), W \neq V)$$

Something similar happens with choice-like pairs of rules for generating possible solutions. They typically have the form of even negative loops, like in the example:

$$\begin{aligned} \mathit{in}(X) &\leftarrow \mathit{vertex}(X), \mathit{not}\ \mathit{out}(X) \\ \mathit{out}(X) &\leftarrow \mathit{vertex}(X), \mathit{not}\ \mathit{in}(X) \\ \bot &\leftarrow \mathit{in}(X), \mathit{in}(Y), X \neq Y, \mathit{not}\ \mathit{edge}(X, Y), \mathit{not}\ \mathit{edge}(Y, X) \end{aligned}$$

[1] Grounders like `lparse` allow avoiding the repetitive definition of variable sorts by specifying general sorted variables with macros of the form `#domain action(A).`

intended for generating a *clique*[2] in terms of predicate $in(X)$. It seems clear that predicate $out(X)$ is auxiliary and thus its use should be limited to this pair of rules (adding other rules with $out(X)$ as a head may change the intended meaning). An alternative to this kind of "generating" loops is using a *cardinality constraint* (as defined in [9]) of the form:

$$\{\ in(X)\ \} \leftarrow vertex(X)$$

whose intuitive meaning is, for each vertex X, pick 0 or 1 atoms in the set $\{\ in(X)\ \}$. As explained in [8] the formula above is (strongly) equivalent to:

$$in(X) \leftarrow vertex(X) \wedge \neg\neg in(X) \tag{4}$$

The informal reading of a formula like $\neg\neg\alpha$ in the body is that "it is consistent to assume α." It is perhaps interesting to note that, as shown in [10], a double negation in the body can be moved (maintaining strong equivalence) to a single negation in the head, as follows:

$$in(X) \vee \neg in(X) \leftarrow vertex(X)$$

and it is well-known [11, 12] that head negations can be ruled out in favour of auxiliary predicates, eventually returning to the same formulation we had with the $out(X)$ predicate. Once again, the interest of the extended syntax is that it can be translated into traditional logic programs while it avoids the explicit use of auxiliary predicates which become hidden in the translation.

3 Overview of Quantified Equilibrium Logic

Following [5], Quantified Equilibirum Logic (QEL) is defined in terms of a models selection criterion for the intermediate logic of Quantified Here-and-There. In the paper, we will deal with a version of this logic dealing with static domains and decidable equality, calling it QHT for short.

Let $\mathcal{L} = \langle C, F, P\rangle$ be a first-order language where C is a set of *constants*, F a set of *functions* and P a set of *predicates*. First-order formulae for $\mathcal{L}$ are built up in the usual way, with the same syntax of classical predicate calculus. As in Intuitionistic Calculus, the formula $\neg\varphi$ will actually stand for $\varphi \rightarrow \bot$. We write $Atoms(C, P)$ to stand for the set of atomic sentences built with predicates in P and constants in C. Similarly, $Terms(C, F)$ denote the set of ground terms built from functions in F and constants in C.

We will adopt a logical writing for logic programming connectives, so that constructions like (α, β), $(not\ \alpha)$ and $(\alpha \leftarrow \beta)$ are respectively written as $(\alpha \wedge \beta)$, $(\neg\alpha)$ and $(\beta \rightarrow \alpha)$. We also adopt lower-case letters for variables and functions, and upper-case for predicates and constants. In this way, a rule like (2) becomes the formula:

$$Holds(f, v, s) \wedge \neg Ab(f, v, a, s) \rightarrow Holds(f, v, do(a, s))$$

[2] A clique is a set of vertices that are pairwise adjacent.

We use boldface letters $\mathbf{x}, \mathbf{y}$ to denote tuples of variables, and similarly $\mathbf{d}$ for tuples of domain elements.

The corresponding semantics for QHT is described as follows.

Definition 1 (QHT-interpretation). *A QHT-*interpretation *for a language $\mathcal{L} = \langle C, F, P\rangle$ is a tuple $\langle (D, \sigma), H, T\rangle$ where:*

1. *D is a nonempty set of constant names identifying each element in the interpretation universe. For simplicity, we take the same name for the constant and the universe element.*
2. *$\sigma : Terms(D \cup C, F) \to D$ assigns a constant in D to any term built with functions in F and constants in the extended set of constants $C \cup D$. It must satisfy: $\sigma(d) = d$ for all $d \in D$.*
3. *H and T are sets of ground atomic sentences such that $H \subseteq T \subseteq Atoms(D, P)$.* □

An interpretation of the form $\langle (D, \sigma), T, T\rangle$ is said to be *total* and can be seen as the classical first-order interpretation $\langle (D, \sigma), T\rangle$. In fact, we will indistinctly use both notations. Furthermore, given any arbitrary $\mathcal{M} = \langle (D, \sigma), H, T\rangle$ we will define a corresponding total (or classical) interpretation $\mathcal{M}_T \stackrel{\text{def}}{=} \langle (D, \sigma), T\rangle$.

Satisfaction of formulas is recursively defined as follows. Given an interpretation $\mathcal{M} = \langle (D, \sigma), H, T\rangle$, the following statements are true:

- $\mathcal{M} \models p(t_1, \ldots, t_n)$ if $p(\sigma(t_1), \ldots, \sigma(t_n)) \in H$.
- $\mathcal{M} \models t_1 = t_2$ if $\sigma(t_1) = \sigma(t_2)$.
- $\mathcal{M} \not\models \bot$.
- $\mathcal{M} \models \alpha \wedge \beta$ if $\mathcal{M} \models \alpha$ and $\mathcal{M} \models \beta$. Disjunction $\vee$ is analogous.
- $\mathcal{M} \models \alpha \to \beta$ if both:
  (i) $\mathcal{M} \not\models \alpha$ or $\mathcal{M} \models \beta$ and
  (ii) $\mathcal{M}_T \models \alpha \to \beta$ in classical logic
- $\mathcal{M} \models \forall x\ \alpha(x)$ if both:
  (i) $\mathcal{M} \models \alpha(d)$, for each $d \in D$ and
  (ii) $\mathcal{M}_T \models \forall x\ \alpha(x)$ in classical logic
- $\mathcal{M} \models \exists x\ \alpha(x)$ if for some $d \in D$, $\mathcal{M} \models \alpha(d)$. □

In the proofs, we will make use of the following property:

Proposition 1. *If $\mathcal{M} \models \varphi$ then $\mathcal{M}_T \models \varphi$.* □

Nonmonotonic entailment is obtained by introducing a models-minimisation criterion. Let us define the following ordering relation among interpretations

Definition 2. *We say that an interpretation $\mathcal{M} = \langle (D, \sigma), H, T\rangle$ is* smaller *than an interpretation $\mathcal{M}' = \langle (D, \sigma), H', T\rangle$, written $\mathcal{M} \preceq \mathcal{M}'$, when $H \subseteq H'$.* □

That is, to be comparable, $\mathcal{M}$ and $\mathcal{M}'$ must only differ in their H component, so that $\mathcal{M} \preceq \mathcal{M}'$ iff $H \subseteq H'$. Notice that, as a consequence, $\mathcal{M} \preceq \mathcal{M}_T$. As usual, we write $\mathcal{M} \prec \mathcal{M}'$ when $\mathcal{M} \preceq \mathcal{M}'$ and $\mathcal{M} \neq \mathcal{M}'$ (that is $H \subset H'$).

We say that $\mathcal{M}$ is a *model* of a theory Γ if $\mathcal{M}$ satisfies all the formulas in Γ. If $\mathcal{M}$ is total, it is easy to check that: $\mathcal{M} \models \Gamma$ iff $\mathcal{M}_T \models \Gamma$ in classical logic. The next definition introduces the idea of minimal models for QHT.

Definition 3 (Equilibrium model). *A total model $\mathcal{M}$ of a theory Γ is an* equilibrium model *if there is no smaller model $\mathcal{M}' \prec \mathcal{M}$ of Γ.* □

Note that an equilibrium model is a total model, i.e., a classical model of Γ. We name *Quantified Equilibrium Logic* (QEL) the logic induced by equilibrium models.

Given an interpretation $\mathcal{M}$ for a given language, and a sublanguage $\mathcal{L}$, we write $\mathcal{M}|_{\mathcal{L}}$ to denote the projection of $\mathcal{M}$ modulo $\mathcal{L}$. We say that two theories Γ_1, Γ_2 for language $\mathcal{L}'$ are *strongly equivalent* with respect to a given sublanguage $\mathcal{L}$ of $\mathcal{L}'$, written $\Gamma_1 \equiv_s^{\mathcal{L}} \Gamma_2$, when for any theory Γ in $\mathcal{L}$, the sets of equilibrium models (modulo $\mathcal{L}$) for $\Gamma_1 \cup \Gamma$ and $\Gamma_2 \cup \Gamma$ coincide. When $\mathcal{L} = \mathcal{L}'$ we just write $\Gamma_1 \equiv_s \Gamma_2$ and, in fact, this has been proved [13] to correspond to the QHT-equivalence of Γ_1 and Γ_2.

A *Herbrand QHT-interpretation* $\mathcal{M} = \langle (D, \sigma), H, T \rangle$ is such that D corresponds to $Terms(C, F)$ and $\sigma = id$, where id is the identity relation. In [13] it was shown that $\mathcal{M}$ is a Herbrand equilibrium model of a logic program Π iff T is a stable model of the (possibly infinite) ground program $gr_D(\Pi)$ obtained by replacing all variables by all terms in D in all possible ways.

4 Bodies with existential quantifiers

In this section we introduce the syntactic extension of logic programs we are interested in. We define a *body* as conjunction of *conditions* being a condition, in its turn, recursively defined as:

i) an atom $P(\mathbf{t})$ where $\mathbf{t}$ is a tuple of terms;
ii) an equality atom $t = t'$ with t, t' terms;
iii) $\exists \mathbf{x}\ (\psi)$ where $\mathbf{x}$ is a tuple of variables and ψ is a body in its turn;
iv) $\neg C$ where C is a condition;

Conditions of the form i) and ii) are called *atoms*. A *literal* is also a condition, with the form of an atom or its negation; the rest of conditions are called *non-literal*. A literal like $\neg(t = t')$ will be abbreviated as $t \neq t'$. Without loss of generality, we can assume that we handle two consecutive negations at most, since $\neg\neg\neg C \leftrightarrow \neg C$ is a QHT-tautology. Conditions beginning (resp. not beginning) with $\neg$ are said to be *negative* (resp. *positive*). Given a body B, we define its *positive* (resp. *negative*) *part*, B^+ (resp. B^-) as the conjunction of positive (resp. negative) conditions in B. We assume that $\exists x_1 \ldots x_n\ \psi$ is a shorthand notation for $\exists x_1 \ldots \exists x_n\ \psi$.

A *rule* is an expression like $B \rightarrow Hd$ where Hd is a (possibly empty) disjunction of atoms (called the rule *head*) and B is a body. We assume that an empty disjunction corresponds to $\bot$. All free variables in a rule are implicitly universally quantified. The following are examples of rules:

$$P(x) \wedge \neg\neg Q(x) \wedge \neg\exists y\ \big(R(x,y) \wedge \exists z\ \neg R(y,z)\big) \rightarrow S(x) \vee R(x,x) \quad (5)$$

$$Person(x) \wedge \neg\exists y(Parent(x,y) \wedge \neg\exists z\ Married(y,z)) \rightarrow Happy(x) \quad (6)$$

$$Vertex(x) \wedge \neg\neg In(x) \rightarrow In(x) \quad (7)$$

$$Vertex(x) \wedge \neg\exists y(Edge(x,y) \wedge \exists z(Edge(y,z))) \rightarrow P(x) \quad (8)$$

Rules (6) and (7) are just different ways of writing (1) and (4) respectively. Rule (8) would intuitively mean, for instance, that $P(x)$ holds for any vertex x that is terminal or exclusively connected to terminal nodes. A rule is said to be *normal* if Hd just contains one atom. If $Hd = \bot$ the rule receives the name of *constraint*. A rule is said to be *regular* if its body is a conjunction of literals (i.e. it does not contain double negations or existential quantifiers).

A set of rules of the general form above will be called a *logic program with existential quantifiers in the body* or $\exists$-logic program, for short. A program is said to be *normal* when all its rules are normal. The same applies for *regular* program.

5 A translation into regular logic programs

The translation of a rule $r : B \to Hd$ into a regular logic program r^* will consist in recursively translating all the negative conditions in the rule body B *with respect to its positive part* B^+. This will possibly generate a set of additional rules dealing with new auxiliary predicates.

Definition 4 (Translation of conditions). *We define the translation of a condition C with respect to a positive body B^+ as a pair $\langle C^\bullet, \Pi(C, B^+)\rangle$ where $C^\bullet$ is a formula and $\Pi(C, B^+)$ a set of rules.*

1. *If C is a literal or has the form $\exists \mathbf{x}\ \alpha(\mathbf{x})$ then $C^\bullet = C$, $\Pi(C, B^+) = \emptyset$.*
2. *Otherwise, the condition has the form $C = \neg\alpha(\mathbf{x})$ being $\mathbf{x}$ the free variables in C. Then $C^\bullet = \neg Aux(\mathbf{x})$ and $\Pi(C, B^+) = (B^+ \wedge \alpha(\mathbf{x}) \to Aux(\mathbf{x}))^*$ where Aux is a new fresh predicate and * is the translation of rules in Definition 5.*

□

The translation of a conjunction of conditions D with respect to a positive body B^+ is defined as expected $\langle D^\bullet, \Pi(D, B^+)\rangle$ where $D^\bullet$ is the conjunction of all $C^\bullet$ for each C in D, and $\Pi(D, B^+)$ the union of all rules $\Pi(C, B^+)$ for each C in D.

Definition 5 (Translation of a rule). *The translation of a rule r, written r^* is done in two steps:*

i) *We begin replacing all the positive conditions $\exists \mathbf{x}\ \varphi$ in the body of r by $\varphi[\mathbf{x}/\mathbf{y}]$ being $\mathbf{y}$ a tuple of new fresh variables[3] and repeat this step until no condition of this form is left. Let $B \to Hd$ denote the resulting rule.*

ii) *We then obtain the set of rules:*

$$r^* \stackrel{\text{def}}{=} \{Hd \leftarrow B^+ \wedge (B^-)^\bullet\} \cup \Pi(B^-, B^+)$$

□

[3] The introduction of new variables $\mathbf{y}$ can be omitted when $\mathbf{x}$ does not occur free in the rest of the rule.

The translation of an $\exists$-logic program Π is denoted Π^* and corresponds to the logic program $\bigcup_{r\in\Pi} r^*$ as expected. As an example of translation, consider the rule $r_1 = (6)$. We would have:

$$r_1^* = \{Person(x) \wedge \neg Aux_1(x) \rightarrow Happy(x)\} \cup \Pi(B(r_1)^-, Person(x))$$

where $B(r_1)^- = \neg\exists y(Parent(x,y) \wedge \neg\exists z(Married(y,z)))$ and so, $\Pi(B(r_1)^-, Person(x))$ contains the translation of the rule:

$$Person(x) \wedge \exists y(Parent(x,y) \wedge \neg\exists z(Married(y,z))) \rightarrow Aux_1(x)$$

We remove the positive existential quantifier[4] to obtain r_2:

$$Parent(x,y) \wedge \neg\exists z(Married(y,z)) \wedge Person(x) \rightarrow Aux_1(x)$$

and now

$$\begin{aligned} r_2^* = \{&Parent(x,y) \wedge Person(x) \wedge \neg Aux_2(x,y) \rightarrow Aux_1(x)\} \\ &\cup \Pi(B(r_2)^-, Parent(x,y) \wedge Person(x)) \end{aligned}$$

This yields the rule

$$Parent(x,y) \wedge Person(x) \wedge \exists z(Married(y,z)) \rightarrow Aux_2(x,y)$$

in which, again, we would just remove the positive existential quantifier. To sum up, the final complete translation r_1^* would be the (regular) logic program:

$$\begin{aligned} Person(x) \wedge \neg Aux_1(x) &\rightarrow Happy(x) \\ Parent(x,y) \wedge Person(x) \wedge \neg Aux_2(x,y) &\rightarrow Aux_1(x) \\ Parent(x,y) \wedge Person(x) \wedge Married(y,z) &\rightarrow Aux_2(x,y) \end{aligned}$$

We can informally read $Aux_2(x,y)$ as "*y is a married child of x*" and $Aux_1(x)$ as "*x has some single child.*"

It is easy to see that the translation is modular (we translate each rule independently) and that its size is polynomial with respect to the original input.

Proposition 2. *Given a rule r containing A atoms in its body and N subformulas of one of the forms* $(\neg\exists \mathbf{x}\ \alpha)$ *or* $(\neg\neg\alpha)$*, the translation* r^* *contains* $N+1$ *regular rules whose bodies contain at most* $A+N$ *atoms in their body.* □

It might be thought that, as we always have a way of removing positive existential quantifiers, these are unnecessary. However, we must take into account that they are useful when nested in another expression. For instance, rule (8) would be translated as:

$$\begin{aligned} Vertex(x) \wedge \neg Aux(x) &\rightarrow P(x) \\ Vertex(x) \wedge Edge(x,y) \wedge Edge(y,z) &\rightarrow Aux(x) \end{aligned}$$

[4] As y is not free in the rest of the formula, there is no need to change it by a new variable, in this case.

but note the difference with respect to a rule like[5]:

$$Vertex(x) \wedge Vertex(z) \wedge \neg\exists y(Edge(x,y) \wedge Edge(y,z)) \rightarrow P(x)$$

whose meaning is drastically different (z is now implicitly universally quantified). This is reflected in the corresponding translation:

$$Vertex(x) \wedge Vertex(z) \wedge \neg Aux(x,z) \rightarrow P(x)$$
$$Vertex(x) \wedge Vertex(z) \wedge Edge(x,y) \wedge Edge(y,z) \rightarrow Aux(x,z)$$

Note how $Aux(x,z)$ depends also on z now, capturing the meaning "there is an intermediate node between x and z." In this way, $P(x)$ would be true, for instance, if we just had any unrelated z without incoming edges.

As an example with double negation, it can be easily checked that the translation of rule (7) becomes the program:

$$Vertex(x) \wedge \neg Aux(x) \rightarrow In(x)$$
$$Vertex(x) \wedge \neg In(x) \rightarrow Aux(x)$$

The intuition behind a double negation of an existential operator can be understood with the following example. Consider the rule:

$$\neg\neg\exists x(Vertex(x) \wedge Marked(x)) \rightarrow P$$

expressing that P holds when *it is consistent to assume* that there exists a marked vertex. The tranlsation would be the program Π_1:

$$\neg Aux_1 \rightarrow P \qquad\qquad Vertex(x) \wedge Marked(x) \rightarrow Aux_2$$
$$\neg Aux_2 \rightarrow Aux_1$$

so that, Aux_2 means "there exists a marked vertex", Aux_1 that "there does not exist a marked vertex" (i.e., all vertices are marked) and so P would mean "I cannot prove there does not exist a marked vertex." It is perhaps worth to observe the difference with respect to program Π_2 just consisting of:

$$Vertex(x) \wedge Marked(x) \rightarrow P$$

where P checks the existence of a marked node. Programs Π_1 and Π_2 are not strongly equivalent (modulo non-auxiliary predicates). For instance, if we consider the addition of program Π, consisting of the facts $Vertex(1)$ and $Vertex(2)\}$ plus the rules

$$P \rightarrow Marked(1) \qquad\qquad Marked(1) \rightarrow P$$

program $\Pi_2 \cup \Pi$ has one stable model $\{Vertex(1), Vertex(2)\}$ whereas $\Pi_1 \cup \Pi$ has a second stable model $\{Vertex(1), Vertex(2), Marked(1), P\}$ (modulo non-auxiliary predicates).

[5] We included in this example an extra condition $Vertex(z)$ just to keep handling a *safe* rule, something we will discuss later.

5.1 Proof of correctness

To prove our main result, we will use several QHT valid equivalences (many of them already commented in [14]) and introduce several lemmas. For instance, we will frequently make use of the following QHT valid formula (see [14])

$$\alpha \rightarrow (\beta \rightarrow \gamma) \leftrightarrow (\alpha \wedge \beta \rightarrow \gamma) \tag{9}$$

Similarly, the following is a QHT-theorem:

$$\alpha \wedge \neg(\alpha \wedge \beta) \leftrightarrow \alpha \wedge \neg\beta \tag{10}$$

whose proof can be obtained from transformations in [10, 14].

Lemma 1. *Let $\mathcal{M}$ be an equilibrium model of Γ, and $\mathcal{M} \models \alpha$. Then $\mathcal{M}$ is an equilibrium model of $\Gamma \cup \{\alpha\}$.* □

Theorem 1 (Equivalent subformula replacement). *Given the equivalence:*

$$\forall \mathbf{x}(\alpha(\mathbf{x}) \leftrightarrow \beta(\mathbf{x})) \tag{11}$$

where $\mathbf{x}$ is the set of free variables in α or β, and a given formula γ containing a subformula $\alpha(\mathbf{t})$, then (11) implies:

$$\gamma \leftrightarrow \gamma[\alpha(\mathbf{t})/\beta(\mathbf{t})]$$

□

Theorem 2 (Defined predicate removal). *Let Γ_1 be a theory for language $\mathcal{L}$, α a formula in that signature and Aux a predicate not in $\mathcal{L}$. If Γ_2 is Γ_1 plus*

$$\forall \mathbf{x}(Aux(x) \leftrightarrow \alpha(\mathbf{x})) \tag{12}$$

then $\Gamma_1 \equiv_s^{\mathcal{L}} \Gamma_2$. □

Lemma 2. *Let $\mathcal{M}_1 = \langle (D, \sigma), H, T \rangle$ be a model of the formulas*

$$\forall \mathbf{x}(\alpha(\mathbf{x}) \rightarrow Aux(\mathbf{x})) \tag{13}$$

$$\forall \mathbf{x}(\neg Aux(\mathbf{x}) \rightarrow \beta(\mathbf{x})) \tag{14}$$

where α and β do not contain predicate Aux, and let $\mathcal{M}_2 = \langle (D, \sigma), H', T \rangle$ be such that $H \backslash H' = \{Aux(\mathbf{d}) \mid \mathbf{d} \in \mathcal{D}\}$ for some non-empty set of tuples of domain elements $\mathcal{D}$ satisfying $\mathcal{M}_1 \models Aux(\mathbf{d})$ and $\mathcal{M}_1 \not\models \alpha(\mathbf{d})$. Then $\mathcal{M}_2 \models (13) \cup (14)$. □

Theorem 3. *Let $\mathcal{L}$ denote a signature not containing predicate Aux, and let $\alpha(\mathbf{x}), \beta(\mathbf{x})$ be a pair of formulas for $\mathcal{L}$. Given $\Gamma_1 = (13) \cup (14)$ and Γ_2 consisting of Γ_1 plus:*

$$\forall \mathbf{x}(Aux(\mathbf{x}) \rightarrow \alpha(\mathbf{x})) \tag{15}$$

then $\Gamma_1 \equiv_s^{\mathcal{L}} \Gamma_2$. □

Theorem 4. *Let Γ_1 be a theory consisting of the single formula*

$$\forall \mathbf{x}\big(\ \alpha(\mathbf{x}) \wedge \neg\beta(\mathbf{x}) \rightarrow \gamma(\mathbf{x})\ \big) \tag{16}$$

for language $\mathcal{L}$, being $\mathbf{x}$ a tuple with all the variables that occur free in the antecedent or in the consequent. Then $\Gamma_1 \equiv_s^{\mathcal{L}} \Gamma_2$ where Γ_2 is the pair of formulas:

$$\forall \mathbf{x}\big(\ \alpha(\mathbf{x}) \wedge \neg Aux(\mathbf{x}) \rightarrow \gamma(\mathbf{x})\ \big) \tag{17}$$

$$\forall \mathbf{x}\big(\ \alpha(\mathbf{x}) \wedge \beta(\mathbf{x}) \rightarrow Aux(\mathbf{x})\ \big) \tag{18}$$

and $Aux(\mathbf{x})$ is a fresh auxiliary predicate not included in $\mathcal{L}$. □

Lemma 3. *Let x be a variable that does not occur free in β. Then, the following is a QHT-tautology:*

$$(\exists x\ \alpha(x) \rightarrow \beta) \leftrightarrow \forall x(\alpha(x) \rightarrow \beta) \tag{19}$$

□

Theorem 5 (Main result). *Let Π be an $\exists$-logic program for language $\mathcal{L}$. Then $\Pi \equiv_s^{\mathcal{L}} \Pi^*$.* □

Theorem 6. *If Π is a disjunctive (resp. normal) $\exists$-logic program then Π^* is a disjunctive (resp. normal) regular logic program.* □

The reason for making the definition of new auxiliary predicates depend on the positive body of the original rule has to do with the following property, that will guarantee a correct grounding of the program resulting from the translation.

Definition 6 (Restricted variable). *A variable X is said to be* restricted *in a conjunction of literals β by a positive literal A in β when X occurs in A and one of the following holds:*

1. *A has the form $p(\mathbf{t})$;*
2. *A has the form $X = Y$ or $Y = X$ and, in its turn, Y is restricted by a different positive literal A' in β.*

We just say that X is restricted *in β if it is restricted by some A in β.* □

Definition 7 (Safe rule). *A rule $r : B \rightarrow Hd$ is said to be safe when both:*

a) Any free variable occurring in r also occurs free and restricted in B.
b) For any condition $\exists x\ \varphi$ in B, x occurs free and restricted in φ. □

For instance, rule (6) is safe: its only free variable x occurs in the positive body $Person(x)$. In fact, all the rules we used in the previous sections are safe. However, rules like:

$$\neg\neg Mark(x) \rightarrow Mark(x)$$
$$\exists y\ Q(y) \rightarrow P(x)$$
$$\exists x\ \neg P(x) \rightarrow A$$

are not safe. Notice that, for regular programs (i.e. those exclusively containing literal conditions) only case *a)* of Definition 7 is applicable and, in fact, this coincides with the usual concept of safe rule in ASP.

Theorem 7. *If Π is safe then Π^* is safe.* □

6 Related work

As commented in the Introduction, this work is directly related to the recently introduced language RASPL-1 [8]. In fact, that language is defined in terms of a translation into first order sentences that fit into the syntax extension we study here (existential quantifiers and double negations in the body). To put an example (extracted from [8]), the RASPL-1 program:

$$\{q(x)\} \leftarrow p(x) \qquad\qquad \bot \leftarrow \{x : q(x)\}\ 1$$

that can be read "give the choice for $q(x)$ per each $p(x)$ but do not take just 0 or 1 atoms for q", actually corresponds to the $\exists$-logic program:

$$p(x) \wedge \neg\neg q(x) \rightarrow q(x) \qquad\qquad \neg\exists xy(q(x) \wedge q(y) \wedge x \neq y) \rightarrow \bot$$

In fact, it is always possible to represent an existential quantifier in the body $\exists x\ \alpha(x)$ using the RASPL-1 construct $1\ \{x : \alpha(x)\}$ provided that $\alpha(x)$ is some literal. In this sense, one of the contributions of this paper is the possibility of recursively nesting quantifiers or double negations inside them. Notice that this is quite comfortable, for instance, in rules like (6) or (8). On the other hand, we provide here a translation of these $\exists$-logic programs into regular logic programs with new hidden auxiliary predicates, preserving strong equivalence modulo the original language.

The use of $\exists$-logic programs was actually forwarded in [7] where an extension of QEL for dealing with partial functions was introduced. The paper considered a syntactic subclass of logic programs with partial functions, providing a translation that removed functions in favour of predicates with an extra parameter (this transformation is usually called *flattening*). The result of this translation, however, fell in the class of $\exists$-logic programs, dealing with negations in the head (i.e. double negations in the body) and possibly nested existential quantifiers in the body. The main result of the current paper was conjectured in [7].

A less related approach that has also considered the use of body quantifiers is [15], although the semantics was only defined for stratified programs.

7 Conclusions

We have presented an extension of logic programming that allows dealing with (possibly nested) existential quantifiers and double negations in the rule bodies. We have shown how this new syntactic class captures several typical representation problems in ASP allowing a more compact and readable formulation and avoiding the use of auxiliary predicates. In fact, we presented a translation that reduces this new syntax to that of regular logic programs by automatically generating these auxiliary predicates, which are kept hidden to avoid programmer's errors.

Several open topics are left for future work. For instance, the direct use of universal quantifiers is not so straightforward. To understand why, notice that

a rule like $\forall x(P(x) \rightarrow Q(x)) \rightarrow H$ is *not* strongly equivalent to $\neg\exists x(P(x) \wedge \neg Q(x)) \rightarrow H$ and its behaviour is related to nested implications, something that in principle is not so clear (see the translation of a nested implication in [14]). With the current formalism, we can just deal with a limited version of $\forall$ since $\neg\exists x(P(x) \wedge \neg Q(x))$ is strongly equivalent to $\neg\neg\forall x(P(x) \rightarrow Q(x))$. Another interesting topic for future study is the use of existential quantifiers in the rule heads.

References

1. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. of the 5th Intl. Conf. on Logic Programming. (1988) 1070–1080
2. Pearce, D., Valverde, A.: Towards a first order equilibrium logic for nonmonotonic reasoning. In: Proc. of the 9th European Conf. on Logics in AI (JELIA'04). (2004) 147–160
3. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI'07). (2004) 372–379
4. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Proc. of the 9th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). (2007) 188–200
5. Pearce, D., Valverde, A.: Quantified equilibrium logic and foundations for answer set programming. In: Proc. of the 24th Intl. Conf. on Logic Programming (ICLP'08). (2008) 547–560
6. Lee, J., Lifschitz, V., Palla, R.: Safe formulas in the general theory of stable models (preliminary report). In: Proc. of the 24th Intl. Conf. on Logic Programming (ICLP'08). (2008) 672–676
7. Cabalar, P.: Partial functions and equality in answer set programming. In: Proc. of the 24th Intl. Conf. on Logic Programming (ICLP'08). (2008) 392–406
8. Lee, J., Lifschitz, V., Palla, R.: A reductive semantics for counting and choice in answer set programming. In: Proc. of the 23rd AAAI Conference on Artificial Intelligence. (2008) 472–479
9. Syrjänen, T.: Cardinality constraint programs. In: Proc. of the 9th European Conf. on Logics in Artificial Intelligence (JELIA'04). (2004) 187–199
10. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Ann. Math. Artif. Intell. **25**(3-4) (1999) 369–389
11. Inoue, K., Sakama, C.: Negation as failure in the head. J. Log. Program. **35**(1) (1998) 39–78
12. Janhunen, T.: On the effect of default negation on the expressiveness of disjunctive rules. In: Proc. of the 6th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning. (2001) 93–106
13. Pearce, D., Valverde, A.: Quantified equilibrium logic and the first order logic of here-and-there. Technical Report MA-06-02, University of Málaga, Spain (2006)
14. Cabalar, P., Pearce, D., Valverde, A.: Reducing propositional theories in equilibrium logic to logic programs. In: 12th Portuguese Conference on Artificial Intelligence (EPIA 2005). (2005) 4–17
15. Eiter, T., Gottlob, G., Veith, H.: Modular logic programming and generalized quantifiers. In: Proc. of the 4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'97). (1997) 290–309

Appendix. Proofs

Proof (Proposition 2). It is easy to see that, excepting for the first step, in which the original rule r is considered, each time we introduce a new rule is for univoquely defining an auxiliary predicate $Aux(\mathbf{x})$ that corresponds to one of the subexpressions of the form $\neg\exists\mathbf{x}\ \alpha$ or $\neg\neg\alpha$ that occurred in r. So, the total number of rules is $N+1$. As for the body size of each rule, we always go keeping a (usually strict) subset of the original number of atoms A occurring in r, plus additional literals $\neg Aux(\mathbf{x})$ corresponding to replaced conditions of the form $\neg\exists\mathbf{x}\ \alpha$ or $\neg\neg\alpha$. As a result, we get the upper bound $N+A$. □

Proof (Lemma 1). Obviously, $\mathcal{M} \models \Gamma \cup \{\alpha\}$. There cannot be a smaller model $\mathcal{M}' \prec \mathcal{M}$ for $\Gamma \cup \{\alpha\}$, because it would also be a model of Γ and this would contradict minimality of $\mathcal{M}$ for Γ. □

Proof (Theorem 1). It is easy to check that, given any tuple $\mathbf{d}$ of domain elements and any model $\mathcal{M} = \langle (D,\sigma), H, T\rangle$, $\mathcal{M} \models (11)$ implies that:

1. $\mathcal{M} \models \alpha(\mathbf{d})$ iff $\mathcal{M} \models \beta(\mathbf{d})$
2. $\mathcal{M}_T \models \alpha(\mathbf{d})$ iff $\mathcal{M}_T \models \beta(\mathbf{d})$

Looking at the satisfaction of formulas, this means that for any model of (11), $\alpha(\mathbf{t})$ and $\beta(\mathbf{t})$ for any tuple of terms $\mathbf{t}$ are semantically equivalent and can be interchanged. □

Proof (Theorem 2). Let Γ denote an arbitrary theory for $\mathcal{L}$ and take $\mathcal{M}$ an equilibrium model of $\Gamma \cup \Gamma_1$ and signature $\mathcal{L}$. We will show that there exists an equilibrium model $\mathcal{M}'$ of $\Gamma \cup \Gamma_2$ such that $\mathcal{M}'|_{\mathcal{L}} = \mathcal{M}$. It is clear we can take $\mathcal{M}'$ equal to $\mathcal{M}$ for all predicates in $\mathcal{L}$ and fix the extent of Aux such that $\mathcal{M}' \models Aux(\mathbf{d})$ iff $\mathcal{M} \models \alpha(\mathbf{d})$ for any tuple of elements $\mathbf{d}$. Obviously, by construction, $\mathcal{M}' \models \Gamma \cup \Gamma_2$. It must also be minimal, since any $\mathcal{M}'' \prec \mathcal{M}'$ that $\mathcal{M}'' \models \Gamma \cup \Gamma_2$ is also a model of $\Gamma \cup \Gamma_1$ and this would contradict the minimality of $\mathcal{M}$ for that theory.

For the other direction, take some $\mathcal{M}' = \langle (D,\sigma), T'\rangle$ equilibrium model of $\Gamma \cup \Gamma_2$. Clearly, $\mathcal{M}' \models \Gamma \cup \Gamma_1$ and, since this theory does not contain Aux, its projection $\mathcal{M}'|_{\mathcal{L}} = \mathcal{M} = \langle (D,\sigma), T\rangle$ must also be a model for $\Gamma \cup \Gamma_1$. Take another model of this theory, $\mathcal{M}_2 = \langle (D,\sigma), H, T\rangle$ with $H \subset T$, that is $\mathcal{M}_2 \prec \mathcal{M}$. But then, we can construct $\mathcal{M}'_2 = \langle (D,\sigma), H', T'\rangle$ such that H' consists of H and the set of atoms $Aux(\mathbf{d})$ for which $\mathcal{M}_2 \models \alpha(\mathbf{d})$. Notice that H' must be a subset of T' because $\mathcal{M}_2 \models \alpha(\mathbf{d})$ implies $\mathcal{M} \models \alpha(\mathbf{d})$ and this implies $\mathcal{M}' \models \alpha(\mathbf{d})$, that together with $\mathcal{M}' \models (12)$ implies and $\mathcal{M}' \models Aux(\mathbf{d})$. But as $H \subset T$ we get $H' \subset T'$ and so $\mathcal{M}'_2 \prec \mathcal{M}'$. On the other hand, by construction of $\mathcal{M}'_2$ together with $\mathcal{M}' \models (12)$, we obtain $\mathcal{M}'_2 \models (12)$. In this way, $\mathcal{M}'_2 \models \Gamma \cup \Gamma_2$ while $\mathcal{M}'_2 \prec \mathcal{M}'$ reaching a contradiction with minimality of $\mathcal{M}'$ for this theory. □

Proof (Lemma 2). Note first that, for any tuple $\mathbf{d} \notin \mathcal{D}$, $\mathcal{M}_1$ and $\mathcal{M}_2$ coincide both for $Aux(\mathbf{d})$, $\alpha(\mathbf{d})$ and $\beta(\mathbf{d})$. Then $\mathcal{M}_1 \models (13)$ and $\mathcal{M}_1 \models (14)$ allow us to conclude $\mathcal{M}_2 \models \alpha(\mathbf{d}) \rightarrow Aux(\mathbf{d})$ and $\mathcal{M}_2 \models \neg Aux(\mathbf{d}) \rightarrow \beta(\mathbf{d})$, respectively. We remain to prove that the same holds for tuples $\mathbf{d} \in \mathcal{D}$. Consider $\mathcal{M}_T = \langle (D, \sigma), T \rangle$, that is, the total model above $\mathcal{M}_1$ and $\mathcal{M}_2$. For any $\mathbf{d} \in \mathcal{D}$, we have $\mathcal{M}_1 \models Aux(\mathbf{d})$ and thus $\mathcal{M}_T \models Aux(\mathbf{d})$, but then $\mathcal{M}_2 \not\models \neg Aux(\mathbf{d})$. On the other hand, $\mathcal{M}_1 \models (14)$ also implies $\mathcal{M}_T \models (14)$ and, in particular, $\mathcal{M}_T \models \neg Aux(\mathbf{d}) \rightarrow \beta(\mathbf{d})$. The latter, together with $\mathcal{M}_2 \not\models \neg Aux(\mathbf{d})$, implies $\mathcal{M}_2 \models \neg Aux(\mathbf{d}) \rightarrow \beta(\mathbf{d})$, for any $\mathbf{d} \in \mathcal{D}$.

Similarly, $\mathcal{M}_1 \models (13)$ implies $\mathcal{M}_T \models (13)$ and, in particular, $\mathcal{M}_T \models \alpha(\mathbf{d}) \rightarrow Aux(\mathbf{d})$ for $\mathbf{d} \in \mathcal{D}$. On the other hand, as $\mathcal{M}_1$ and $\mathcal{M}_2$ do not differ for $\alpha(\mathbf{d})$, we conclude $\mathcal{M}_2 \not\models \alpha(\mathbf{d})$, and thus, $\mathcal{M}_2 \models \alpha(\mathbf{d}) \rightarrow Aux(\mathbf{d})$. □

Proof (Theorem 3). Let Γ denote an arbitrary theory for $\mathcal{L}$ and take $\mathcal{M} = \langle (D, \sigma), T \rangle$ an equilibrium model of $\Gamma \cup \Gamma_1$. For proving that $\mathcal{M}$ is equilibrium model of $\Gamma \cup \Gamma_2$, by Lemma 1, it suffices to show that $\mathcal{M} \models (15)$. Assume this does not hold. As $\mathcal{M}$ is a total model, this just means that for some tuple of domain elements $\mathbf{d}$, $\mathcal{M} \models Aux(\mathbf{d})$ and $\mathcal{M} \not\models \alpha(\mathbf{d})$. Let us take now a model $\mathcal{M}' = \langle (D, \sigma), H, T \rangle$ where H is equal to T excepting that the extension of Aux does not include the tuple $\mathbf{d}$. Notice that $H \subset T$ and $\mathcal{M}' \prec \mathcal{M}$. In fact, we can observe that Lemma 2 is applicable taking $\mathcal{M}_1 = \mathcal{M}$, $\mathcal{M}_2 = \mathcal{M}'$ and $\mathcal{D} = \{\mathbf{d}\}$ to conclude $\mathcal{M}' \models (13) \cup (14)$, i.e., $\mathcal{M}' \models \Gamma_1$. Furthermore, as $\mathcal{M}'$ only differs from $\mathcal{M}$ in Aux, $\mathcal{M}' \models \Gamma$. But this contradicts the minimality of $\mathcal{M}$ as equilibrium model of $\Gamma \cup \Gamma_1$.

For the other direction, let $\mathcal{M}$ be an equilibrium model of $\Gamma \cup \Gamma_2$. Since $\Gamma_1 \subset \Gamma_2$, obviously $\mathcal{M} \models \Gamma \cup \Gamma_1$. We remain to prove that $\mathcal{M}$ is minimal. Suppose we had some other model $\mathcal{M}' \prec \mathcal{M}$ of $\Gamma \cup \Gamma_1$. If $\mathcal{M}' \models (15)$ we would have $\mathcal{M}' \models \Gamma \cup \Gamma_2$ and this would contradict the minimality of $\mathcal{M}$ for that theory. So, assume $\mathcal{M}' \not\models (15)$. Let $\mathcal{D}$ be the set of tuples $\mathbf{d}$ for which $\mathcal{M}' \not\models Aux(\mathbf{d}) \rightarrow \alpha(\mathbf{d})$ (note that this set cannot be empty). As $\mathcal{M} \models (15)$ we must have $\mathcal{M}' \models Aux(\mathbf{d})$ and $\mathcal{M}' \not\models \alpha(\mathbf{d})$ for all $\mathbf{d} \in \mathcal{D}$. Now take $\mathcal{M}''$ equal to $\mathcal{M}'$ excepting that, for all $\mathbf{d} \in \mathcal{D}$, $\mathcal{M}'' \not\models Aux(\mathbf{d})$. We can apply Lemma 2 taking $\mathcal{M}_1 = \mathcal{M}'$, $\mathcal{M}_2 = \mathcal{M}''$ and $\mathcal{D}$ to conclude $\mathcal{M}'' \models (13) \cup (14)$, i.e., $\mathcal{M}'' \models \Gamma_1$. Furthermore, as $\mathcal{M}''$ only differs from $\mathcal{M}'$ in the extent of Aux, we obtain $\mathcal{M}'' \models \Gamma \cup \Gamma_1$. Now, as $\mathcal{M}'' \not\models Aux(\mathbf{d})$ and we have $\mathcal{M} \models (15)$ we conclude $\mathcal{M}'' \models Aux(\mathbf{d}) \rightarrow \alpha(\mathbf{d})$. For tuples $\mathbf{c} \notin \mathcal{D}$ we had $\mathcal{M}' \models Aux(\mathbf{c}) \rightarrow \alpha(\mathbf{c})$ by definition of $\mathcal{D}$, but $\mathcal{M}'$ and $\mathcal{M}''$ coincide in $Aux(\mathbf{c})$ and $\alpha(\mathbf{c})$. As a result, $\mathcal{M}'' \models (15)$ too, and since $\mathcal{M}'' \prec \mathcal{M}$ we obtain a contradiction with minimality of $\mathcal{M}$ for $\Gamma \cup \Gamma_2$. □

Proof (Theorem 4). By (9), the formula (17) is strongly equivalent to:

$$\forall \mathbf{x} \big(\neg Aux(\mathbf{x}) \rightarrow (\alpha(\mathbf{x}) \rightarrow \gamma(\mathbf{x})) \big) \tag{20}$$

so that, we can apply Theorem 3 on Γ_2 to transform the implication in (18) into a double implication:

$$\forall \mathbf{x} \big(\alpha(\mathbf{x}) \wedge \beta(\mathbf{x}) \leftrightarrow Aux(\mathbf{x}) \big) \tag{21}$$

As a result, Γ_2 is strongly equivalent (modulo $\mathcal{L}$) to the theory consisting of (17) and (21). By Theorem 1, this is strongly equivalent, in its turn, to (21) plus:

$$\forall \mathbf{x}\big(\ \alpha(\mathbf{x}) \wedge \neg(\alpha(\mathbf{x}) \wedge \beta(\mathbf{x})) \rightarrow \gamma(\mathbf{x})\ \big) \tag{22}$$

Due to (10), the latter is strongly equivalent to (16). Finally, by Theorem 2, we can remove (21), since it is a definition for predicate Aux which does not belong to $\mathcal{L}$. □

Proof (Lemma 3). As (19) is a classical tautology, we remain to prove that, for any interpretation $\mathcal{M} = \langle (D, \sigma), H, T \rangle$, $\mathcal{M} \models \exists x\ \alpha(x) \rightarrow \beta$ iff $\mathcal{M} \models \forall x(\alpha(x) \rightarrow \beta)$. For the left to right direction, assume $\mathcal{M} \models \exists x\ \alpha(x) \rightarrow \beta$ but $\mathcal{M} \not\models \forall x(\alpha(x) \rightarrow \beta)$. The latter means there exists some element d for which $\mathcal{M} \not\models \alpha(d) \rightarrow \beta$. Since $\mathcal{M} \models \exists x\ \alpha(x) \rightarrow \beta$ we have that $\mathcal{M}_T$ also satisfies that formula and so $\mathcal{M}_T \models \forall x(\alpha(x) \rightarrow \beta)$ since it is a classically equivalent formula. Therefore, the only possibility is $\mathcal{M} \models \alpha(d)$ and $\mathcal{M} \not\models \beta$. But from the former we get $\mathcal{M} \models \exists x\ \alpha(x)$ and this contradicts $\mathcal{M} \models \exists x\ \alpha(x) \rightarrow \beta$.

For the right to left direction, suppose $\mathcal{M} \models \forall x(\alpha(x) \rightarrow \beta)$. As $\mathcal{M}_T$ also satisfies that formula it must also satisfy the classically equivalent formula $\exists x\ \alpha(x) \rightarrow \beta$. We remain to prove that $\mathcal{M} \models \exists x\ \alpha(x)$ implies $\mathcal{M} \models \beta$. Assume that the former holds. Then, for some element d, $\mathcal{M} \models \alpha(d)$. As $\mathcal{M} \models \forall x(\alpha(x) \rightarrow \beta)$, in particular, $\mathcal{M} \models \alpha(d) \rightarrow \beta$, but this together with $\mathcal{M} \models \alpha(d)$ implies $\mathcal{M} \models \beta$. □

Proof (Theorem 5. Main result). We prove the result by induction on the successive application of $\cdot^*$ in each group of newly generated rules. If a rule r is regular it can be easily checked that $r^* = r$ and the result of strong equivalence is straightforward. If r contains a double negation or an existential quantifier, we will show that the two steps in Definition 5 preserve strong equivalence. Step i) is the result of the successive application of Lemma 3, that allows us to remove a positive existential quantifier in the body, provided that the quantified variable does not occur free in the rest of the formula. Notice that this lemma can be applied to a larger body like $\exists x\ \alpha(x) \wedge \gamma \rightarrow \beta$ (again, with x not free in γ) because the latter is QHT-equivalent to $\exists x\ \alpha(x) \rightarrow (\gamma \rightarrow \beta)$. For Step ii), consider any rule $r : B \rightarrow Hd$ with some non-literal negative condition $\neg\beta_1(\mathbf{x})$. We can write r as $B^+(\mathbf{x}) \wedge \neg\beta_1(\mathbf{x}) \wedge B'(\mathbf{x}) \rightarrow Hd(\mathbf{x})$, being $B'(\mathbf{x})$ the rest of conjuncts in the negative body, that is, $B^-(\mathbf{x})$ excepting $\neg\beta_1(\mathbf{x})$. This expression can be equivalently written as $B^+(\mathbf{x}) \wedge \neg\beta_1(\mathbf{x}) \rightarrow (B'(\mathbf{x}) \rightarrow Hd(\mathbf{x}))$ and so, we can apply Theorem 4 taking $\alpha(\mathbf{x})$ to be the positive body $B^+(\mathbf{x})$, and $\gamma(\mathbf{x})$ the implication $B'(\mathbf{x}) \rightarrow Hd(\mathbf{x})$ to conclude that r is strongly equivalent (modulo its original language $\mathcal{L}$) to the conjunction of $B^+(\mathbf{x}) \wedge \neg Aux_1(\mathbf{x}) \wedge B'(\mathbf{x}) \rightarrow Hd(\mathbf{x})$ plus $B^+(\mathbf{x}) \wedge \beta_1(\mathbf{x}) \rightarrow Aux_1(\mathbf{x})$ being Aux_1 a new fresh predicate. We can repeat this step for the rest of non-literal negative conditions in B^- until the original rule becomes $B^+(\mathbf{x}) \wedge \neg Aux_1(\mathbf{x}) \wedge \cdots \wedge \neg Aux_n(\mathbf{x}) \wedge B''(\mathbf{x}) \rightarrow Hd(\mathbf{x})$, i.e., what we called $B^+ \wedge (B^-)^\bullet \rightarrow Hd$ in Definition 5. Finally, the correctness of the

translation of the newly generated rules $B^+(\mathbf{x}) \wedge \beta_i(\mathbf{x}) \rightarrow Aux_i(\mathbf{x})$ follows from the induction hypothesis. Note that termination of this inductive transformation $\cdot^*$ is guaranteed by observing that in each step, we reduce the size of new rule bodies, replacing negative non-literal conditions by smaller expressions. □

Proof (Theorem 6). First, observe that all the rules generated in the translation either repeat one of the original rule heads in Π or just contain one atom $Aux(\mathbf{x})$. Thus, if the original program was disjunctive (resp. normal) then Π^* will be disjunctive (resp. normal). Second, just notice that the translation is recursively repeated until rule bodies exclusively contain literal conditions, so the final program will be a regular logic program in the usual sense. □

Proof (Theorem 7). It suffices to observe that the rules generated in each translation step preserves safety with respect to Definition 7. Assume we start from a safe rule and obtain its translation following the steps in Definition 5. In Step i) of that definition, each time we remove $\exists x \varphi$ and replace it by $\varphi[x/y]$ we are introducing a new free variable y in the rule that must satisfy condition *a)* in Definition 7 to maintain safety. But this is guaranteed because the original rule was safe and so, x occurred free and outside the scope of negation in φ. Therefore, y will occur free and outside the scope of negation in $\varphi[x/y]$, which is part of the resulting rule body. This means that the resulting rule satisfies *a)* in Definition 7 for variable y while the status of the rest of variables in the rule has not changed.

Now, take the rule $r : B \rightarrow Hd$ that results from iterating Step i) which, as we have seen, preserves safety. Notice that r does not contain quantified expressions outside the scope of negation, so that B^+ is just a conjunction of atoms. It can be easily observed that each rule $r' : B^+ \wedge (B^-)^\bullet \rightarrow Hd$ does not introduce new free variables with respect to $B \rightarrow Hd$ (it just replaced any negative condition like $\neg\alpha(\mathbf{x})$ in B^- by a new atom $\neg Aux(\mathbf{x})$) while it maintains the original positive body B^+. So, as the original rule r was safe, all free variables in r' also satisfy condition *a)* in Definition 7, while *b)* is not applicable because r' is regular (its body exclusively consists of literals). Similarly, rules like $r'' : B^+ \wedge \alpha(\mathbf{x}) \rightarrow Aux(\mathbf{x})$ in $\Pi(B^-, B^+)$ do not introduce new free variables with respect to r either, while they maintain the same positive body B^+, so they will satisfy *a)* in Definition 7. On the other hand, any quantified condition like $\exists y\ \varphi$ that occurs in $\alpha(\mathbf{x})$ also occurred in a condition $\neg\alpha(\mathbf{x})$ in r. As r was safe, $\exists y\ \varphi$ will satisfy *b)* in Definition 7, so that rule r'' is safe too. □