

Modeling epistemic knowledge in logic programs with negation as failure

Mircea Preda

Department of Computer Science, University of Craiova
Al. I. Cuza 13, 200585 Craiova, Romania
mirceapreda@central.ucv.ro

Abstract. Representing what an intelligent agent knows or believes can be an important feature for a Web service. Epistemic logic programs proposed by Gelfond [6] are a clear way to reason about what an intelligent agent knows or believes, but the complexity of the reasoning is exponential. The paper proposes a new class of logic programs, which consists from logic programs with two types of negation as failure operators. An answer set like semantics is defined for these programs and it is mathematically proved that this semantics allows representing epistemic knowledge. The new programs are able to deal with information like "know p" or "believe p" without introducing new logic operators as for epistemic logic programs. The semantics for "know" and "believe" can be programmed by using specific patterns of rules. In logic programming, the well founded semantics is often considered as a polynomial approximation of the answer set semantics. In a similar way, it has been developed a polynomial approximation for the answer set semantics of the new class of logic programs. This approximation provides both inferior and superior boundaries of the answers constructed by following the answer set semantics. Moreover, it is mathematically proved that the approximation maintains the epistemic knowledge representation possibilities. The meaning of the "know p" and "believe p" pieces of information can still be captured (programmed) by using only the well founded like semantics. Several knowledge representation examples in the Web services context including some difficult situations like representing the unknown and representing the closed world assumption concludes the paper. For each example are presented both the results provided by the answer set semantics and the results provided by the well founded semantics and it is showed that the essential meaning of example is captured by both semantics.

Key words: Knowledge specification, Modeling techniques, Epistemic logic programs, Answer set semantics, Well founded semantics

1 Introduction

Epistemic logic programs ([6], [10]) are a particular type of logic programs that are able to represent epistemic information about an intelligent agent. Epistemic

information can be important for the service oriented computing paradigm. In the web services world is highly important to understand if two web services can inter-operate by reasoning about their inter-operation policies. Often, these policies are represented by sets of rules and, consequently, logic programming is an appealing mathematical framework to represent and study the properties of the interaction policies. Logic programming provides reasoning methods that allows to a web service to analyze its compatibility with the behavior of another web service. This approach was already discussed in papers like [1], which propose an abductive logical framework for the reasoning processes required by a web service, and [8], which combines web services, intelligent agents and computational logic, for implementing logic-based agents that reason about interaction protocols. But sometimes, finer logical representations are necessary. A web service sometimes needs to reason about what it knows or believes about the inter-operation policies of another web service or about its own interaction policies. Epistemic logic programs are a great formalism for this task but their computational complexity is too high (exponential regarding the number of atoms from an epistemic logic program).

In this paper, we propose a new type of logic programs, the logic programs with two type of negation as failure operators and show that these programs are able to clearly and efficiently represent epistemic information. First, the syntax and the answer set semantics of the logic programs with two negations (LP2N) are defined and it is proved that the answer set semantics allows representing what an intelligent agent knows or believes. After that, a polynomial time approximation for the answer set semantics is constructed and it is proved that the approximation maintains the epistemic capabilities of the answer set semantics. Finally, two examples of knowledge representation using these programs are provided. The examples present a scenario where two web services try to interact and one web service use epistemic information to establish if the second web service satisfies its interaction compatibility criteria.

2 Mathematical background

Definition 1. Let $(L, <)$ be a partial ordered set and $A \subseteq L$. $z \in L$ is an infimum of A if and only if $\forall x \in A, z < x$ and $\forall z' \in L$ with this property $z' < z$. In other words, z is the biggest element from L that is smaller than the elements from A . $y \in L$ is a supremum of A if and only if $\forall x \in A, x < y$ and $\forall y' \in L$ with this property, $y < y'$. y is the smallest element from L that is bigger than the elements of A .

Definition 2. Let us consider a partial ordered set $(L, <)$. $(L, <)$ is a lattice if and only if $\forall x, y \in L$ the set $\{x, y\} \subseteq L$ has an infimum and a supremum. $(L, <)$ is a complete lattice if every subset $A \subseteq L$ has an infimum and a supremum.

Remark 1. A complete lattice cannot be empty. Let $(L, <)$ be a complete lattice and $L = \emptyset$. Then L has no supremum and no infimum and the definition of complete lattice is contradicted.

Definition 3. Let $(L, <)$ a partial ordered set and $f : L \rightarrow L$ a function. f is called monotone if and only if $\forall x, y \in L, x < y$ then $f(x) < f(y)$. f is called antimotone if and only if $\forall x, y \in L, x < y$ then $f(y) < f(x)$.

Theorem 1. (Knaster-Tarski theorem [7]) Let $(L, <)$ be a complete lattice and let $f : L \rightarrow L$ a monotone function. Then the set of the fixpoints of $f(\cdot)$ in L is also a complete lattice.

Remark 2. Since complete lattices cannot be empty, the theorem in particular guarantees the existence of at least one fixed point of $f(\cdot)$, and even the existence of a least (or greatest) fixed point. The smallest fixpoint of $f(\cdot)$ is the smallest element x with the property that $f(x) = x$ or, equivalently, $f(x) < x$. The greatest fixpoint of $f(\cdot)$ is the greatest element x with the property that $f(x) = x$.

Definition 4. A definite logic program Π is a collection of rules

$$C \leftarrow A_1, \dots, A_m$$

where $m \geq 0$ and C, A_1, \dots, A_m are ground atoms. The set of the all atoms from a definite logic program Π is called Herbrand Base and denoted by $HB(\Pi)$. The minimal model of a definite logic program Π , denoted by $m(\Pi)$, is the smallest subset of $HB(\Pi)$ with the property that for every rule $C \leftarrow A_1, \dots, A_m$ if $C \in m(\Pi)$ then $\{A_1, \dots, A_m\} \subseteq m(\Pi)$.

Remark 3. A rule $C \leftarrow A_1, \dots, A_m$ has the meaning: if the facts A_1, \dots, A_m are true then C is also true. The minimal model $m(\Pi)$ represents the minimal set of facts that can be inferred to be true based on the rules from Π .

3 Logic programs with two negation operators

Let us consider two operators of negation as failure: not_1 and not_2 .

Definition 5. Formally, by a logic program with two negation operators (LP2N), we mean a collection(set) of rules that can have two forms:

$$\begin{aligned} C &\leftarrow A_1, \dots, A_m, not_1 B_1, \dots, not_1 B_n \\ C &\leftarrow A_1, \dots, A_m, not_2 B_1, \dots, not_2 B_n \end{aligned}$$

where $m, n \geq 0$, C, A_s, B_s are ground atoms.

The expression to the left of \leftarrow is called the *head* of the rule, while the expression to the right of \leftarrow is called the *body* of the rule.

Definition 6. Let Π be a LP2N. By $ord(\Pi)$, we denote the greatest index of the negation as failure operators that appear in Π . $ord(\Pi) \leq 2$. If Π does not contain negation as failure then $ord(\Pi) = 0$.

The expressions *not B* involving *negation as failure* operator are commonly considered as having the meaning: if *B* cannot be proved to be true based on the rules from *II* then it will be considered to be false. For accuracy, this statement can be rephrased as: *not B* can be supposed to be true if this supposition does not generate a contradiction when it is inserted in the program *II*. A contradiction appears if the program obtained after insertion infers *B*.

Consequently, *negation as failure* can be handled as follows: suppositions are made about the truth or falsity of the all *negation as failure* elements that appear in the program. Following these suppositions, *negation as failure* can be removed from the initial program and a definite logic program is obtained. The minimal model of this definite logic program can confirm our suppositions or not. Transformation of a logic program with *negation as failure* into a definite logic program is described by the *Gelfond-Lifschitz* transformation [5].

LP2Ns use two negation as failure operators: *not₁* and *not₂*. *not₁* is the equivalent of the classical *negation as failure*. A program that includes *not₁* can have several distinct sets of facts that can be inferred in accordance with the suppositions that are made about the truth or falsity of the *not₁* elements. Each set of facts that is supported by the logic program describes a possible state of the world. *not₂* operators allow to make suppositions about the atoms that appear in the all possible states of the world, states that are described by the *not₁* operators. Suppositions about the truth or falsity of the *not₁* and *not₂* elements must be made at different stages and the procedure used to eliminate *not₁* and *not₂* from programs is described by the following proposition.

Definition 7. (an extension of Gelfond-Lifschitz transformation [5]) Let *II* be a *LP2N* and $\text{ord}(II) = k \geq 1$. For any set *S* of literals, we denote by *II^S* the *LP2N* obtained from *II* by deleting

- (I) each rule that has a formula $\text{not}_k B$ in its body with $B \in S$
- (II) all formulas of the form $\text{not}_k B$ in the bodies of the remaining rules.

$$\text{ord}(II^S) \leq k - 1.$$

Definition 8. Let *II* be a *LP2N* such that $\text{ord}(II) = 0$. The answer set of *II* is the smallest (in the sense of the set-theoretic inclusion) subset $S \subseteq \text{HB}(II)$ such that for any rule $C \leftarrow A_1, \dots, A_m$ from *II* if $\{A_1, \dots, A_m\} \subseteq S$ then $C \in S$. Let *II* be a *LP2N* and $\text{ord}(II) \geq 1$. We say that $S \subseteq \text{HB}(II)$ is an answer set of *II* if and only if

$$S = \bigcap_{S' \in \text{AS}(II^S)} S'$$

where, for any *LP2N* *II*, $\text{AS}(II)$ represents the family of the all answer sets of *II*.

An answer set should be viewed as a possible state of the world. Let *II* be an *LP2N* and $S \in \text{AS}(II)$ an answer set. The answer of the pair (*II*, *S*) to an atom query *q* is *Yes* under answer set semantics if $q \in S$ and *No* if $q \notin S$. Regarding to a *LP2N* *II*, an atom *a* can be considered known if $a \in \bigcap_{S \in \text{AS}(II)} S$ (*a* appears

in the all possible states of the world) and a can be believed if $a \in \bigcup_{S \in AS(\Pi)} S$ (a appears in at least one possible state of the world).

Example 1. Let us consider the LP2N Π including the following rules:

$$\begin{aligned} a &\leftarrow not_1 b \\ b &\leftarrow not_1 a \\ c &\leftarrow not_2 a \end{aligned}$$

The rule $c \leftarrow not_2 a$ has the meaning: c is true if the supposition that a is false is not contradicted by the all possible states of world for the program that results by making this supposition. The first two rules represent the disjunction $a \vee b \leftarrow ([9])$. Consequently, there are possible states of the world where a is false. Indeed, $S = \{c\}$ is the only answer set of Π . Π^S is the program

$$\begin{aligned} a &\leftarrow not_1 b \\ b &\leftarrow not_1 a \\ c &\leftarrow \end{aligned}$$

and has two answer sets: $\{a, c\}$ and $\{b, c\}$. Their intersection is $S = \{c\}$.

Proposition 1. Let Π be a LP2N with $ord(\Pi) = 1$ and $a \in HB(\Pi)$. We build the LP2N Π' adding to Π the rules

$$\begin{aligned} p &\leftarrow not_1 a \\ q &\leftarrow not_2 p \end{aligned}$$

where p, q are new atoms, $p, q \notin HB(\Pi)$.

In these conditions, the following equivalences are true:

- (I) $\exists S \in AS(\Pi)$ such that $a \in S \iff \Pi' \models q$ (i.e. Π' 's answer to the query q is yes).
- (II) $\forall S \in AS(\Pi)$, $a \notin S \iff \Pi' \models p$.

Proof: We will prove that Q is an answer set of Π' if and only if

$$Q = \begin{cases} \bigcap_{S \in AS(\Pi)} S \cup \{p\} & \text{if } a \notin \bigcup_{S \in AS(\Pi)} S \\ \bigcap_{S \in AS(\Pi)} S \cup \{q\} & \text{if } a \in \bigcup_{S \in AS(\Pi)} S \end{cases}.$$

" \Rightarrow " Consider an answer set $Q \in AS(\Pi')$, $Q = \bigcap_{S' \in AS(\Pi'^Q)} S'$. There are two cases.

Case 1. $p \in Q$. Then $\Pi'^Q = \Pi \cup \{p \leftarrow not_1 a\}$. The following equivalence is obvious: $S \in AS(\Pi)$ iff $S' \in AS(\Pi'^Q)$ where $S' = \begin{cases} S & \text{if } a \in S \\ S \cup \{p\} & \text{if } a \notin S \end{cases}$. Therefore,

$$\bigcap_{S' \in AS(\Pi'^Q)} S' = \begin{cases} \bigcap_{S \in AS(\Pi)} S & \text{if } a \in \bigcup_{S \in AS(\Pi)} S \\ \bigcap_{S \in AS(\Pi)} S \cup \{p\} & \text{if } a \notin \bigcup_{S \in AS(\Pi)} S \end{cases}$$

But $p \in Q$, therefore $a \notin \bigcup_{S \in AS(\Pi)} S$.

Case 2. $p \notin Q$. Then $\Pi'^Q = \Pi \cup \{p \leftarrow \text{not}_1 a, q \leftarrow\}$. The following equivalence is obvious: $S \in AS(\Pi)$ iff $S' \in AS(\Pi'^Q)$ where

$S' = \begin{cases} S \cup \{q\} & \text{if } a \in S \\ S \cup \{p, q\} & \text{if } a \notin S \end{cases}$. Therefore

$$\bigcap_{S' \in AS(\Pi'^Q)} S' = \begin{cases} \bigcap_{S \in AS(\Pi)} S \cup \{q\} & \text{if } a \in \bigcup_{S \in AS(\Pi)} S \\ \bigcap_{S \in AS(\Pi)} S \cup \{p, q\} & \text{if } a \notin \bigcup_{S \in AS(\Pi)} S \end{cases}$$

But $p \notin Q$, therefore $a \in \bigcup_{S \in AS(\Pi)} S$.

" \Leftarrow " Case 1. Consider $Q = \bigcap_{S \in AS(\Pi)} S \cup \{p\}$ and $a \notin \bigcup_{S \in AS(\Pi)} S$. Then $\Pi'^Q = \Pi \cup \{p \leftarrow \text{not}_1 a\}$. We obtain that $\bigcap_{S' \in AS(\Pi'^Q)} S' = \bigcap_{S \in AS(\Pi)} S \cup \{p\} = Q$, therefore $Q \in AS(\Pi')$.

Case 2. Consider $Q = \bigcap_{S \in AS(\Pi)} S \cup \{q\}$ and $a \in \bigcup_{S \in AS(\Pi)} S$. Then $\Pi'^Q = \Pi \cup \{p \leftarrow \text{not}_1 a, q \leftarrow\}$. We obtain that $\bigcap_{S' \in AS(\Pi'^Q)} S' = \bigcap_{S \in AS(\Pi)} S \cup \{q\} = Q$, therefore $Q \in AS(\Pi')$.

Remark 4. Π' has only one answer set.

Remark 5. It is natural to say that an intelligent agent, represented by a set of premises Π , may believe that an atom a is true if and only a occurs in at least one answer set of Π . The operator M (believe operator) defined for epistemic logic programs [6] can be incorporated in this way into LP2Ns. The newly introduced atoms p and q indicate us if a occurs in at least one answer set of Π or not.

Proposition 2. *Let Π be a LP2N with $\text{ord}(\Pi) = 1$ and $a \in \text{HB}(\Pi)$. We build the program Π' adding to Π the rule*

$$p \leftarrow \text{not}_2 a$$

where p is a new atom, $p \notin \text{HB}(\Pi)$. In these conditions, the following equivalences are true:

- (I) $a \in S, \forall S \in AS(\Pi) \iff \Pi' \models a$
- (II) $\exists S \in AS(\Pi)$ such that $a \notin S \iff \Pi' \models p$.

Proof: The proof is based on the equivalence: Q is an answer set of Π' if and only if

$$Q = \begin{cases} \bigcap_{S \in AS(\Pi)} S & \text{if } a \in \bigcap_{S \in AS(\Pi)} S \\ \bigcap_{S \in AS(\Pi)} S \cup \{p\} & \text{if } a \notin \bigcap_{S \in AS(\Pi)} S \end{cases}$$

This proposition allows to incorporate the operator K (know operator) defined for epistemic logic programs into $LP2N$ s. It is natural to say that an intelligent agent, with a set of premises Π , knows that an atom a is true if and only if a occurs in all answer sets of Π . The newly introduced atom p indicates if a does not occur in all answer sets of Π .

4 An approximation for the answer set semantics

The answer set semantics provides us intuitive answers but in practice its exponential complexity is a major drawback.

Let us consider the function $f_\Pi : HB(\Pi) \rightarrow HB(\Pi)$

$$f_\Pi(S) = \begin{cases} m(\Pi) & \text{if } ord(\Pi) = 0 \\ \inf\{S' \mid S' = f_{\Pi^S}(S')\} & \text{otherwise} \end{cases} ,$$

where Π is a $LP2N$, $m(\Pi)$ is the minimal model of the $LP2N$ Π and $\inf\{S' \mid S' = f_{\Pi^S}(S')\} = \bigcap_{S' = f_{\Pi^S}(S')} S'$.

A $LP2N$ Π can be divided in three parts: Π_0 is the set of rules of Π that do not contain negation as failure, Π_1 is the set of rules of Π that contain the not_1 negation as failure operator and, similarly, Π_2 includes the rules that contain the not_2 operator.

Definition 9. A $LP2N$ Π is named stratified if and only if the following condition hold:

$$head(\Pi_2) \cap body(\Pi_1 \cup \Pi_0^1) = \emptyset$$

where $head(\Pi)$ is the set of the atoms which occur in the heads of rules from Π , $body(\Pi)$ is the set of the atoms from the bodies of rules from Π and $\Pi_0^1 \subseteq \Pi_0$ is the smallest subset of rules of Π_0 which satisfies the following properties:

- it contains all rules of Π_0 with the property that their heads occur in a body of rule from Π_1
- if the head of a rule from Π_0 occurs in the body of a rule from Π_0^1 then this rule belongs to Π_0^1 .

Lemma 1. Let Π be a stratified $LP2N$ with $ord(\Pi) = k \leq 2$. Then, the application $f_\Pi(\cdot)$ is anti-monotone.

Proof: Let $S_1 \subseteq S_2 \subseteq HB(\Pi)$. It can be proved that $f_\Pi(S_1) \supseteq f_\Pi(S_2)$.

If $ord(\Pi) = 0$ then $f_\Pi(S_1) = f_\Pi(S_2) = m(\Pi)$.

If $ord(\Pi) = 1$ then $\Pi^{S_1} \supseteq \Pi^{S_2}$, $ord(\Pi^{S_1}) = ord(\Pi^{S_2}) = 0$, $m(\Pi^{S_1}) \supseteq m(\Pi^{S_2})$ and, consequently, $f_\Pi(S_1) \supseteq f_\Pi(S_2)$.

If $ord(\Pi) = 2$ then $\Pi^{S_1} \supseteq \Pi^{S_2}$, $ord(\Pi^{S_1}) = ord(\Pi^{S_2}) \leq 1$. The following relations are true:

$$\begin{aligned} \Pi^{S_1} &= \Pi_2^{S_1} \cup (\Pi_1 \cup \Pi_0), \\ \Pi^{S_2} &= \Pi_2^{S_2} \cup (\Pi_1 \cup \Pi_0), \\ \Pi_2^{S_1} &\supseteq \Pi_2^{S_2}, \\ \Pi^{S_1} \setminus \Pi^{S_2} &= \Pi_2^{S_1} \setminus \Pi_2^{S_2}, \\ 0 &= ord(\Pi^{S_1} \setminus \Pi^{S_2}). \end{aligned}$$

Due to stratification,

$$\text{head}(\Pi^{S_1} \setminus \Pi^{S_2}) \cap (\Pi_1 \cup \Pi_0^1) = \emptyset. \quad (1)$$

Let S' be a fixpoint of the function $f_{\Pi^{S_1}}(\cdot)$, $f_{\Pi^{S_1}}(S') = S'$. There are two cases.

Case 1. $\text{ord}(\Pi^{S_1}) = 0$. Then $\text{ord}(\Pi^{S_2}) = 0$. $f_{\Pi^{S_1}}(S') = m(\Pi^{S_1})$, so $S' = m(\Pi^{S_1})$. There is $S'' = m(\Pi^{S_2})$ such that $S'' \subseteq S'$ and $S'' = f_{\Pi^{S_2}}(S'')$.

Case 2. $\text{ord}(\Pi^{S_1}) = 1$. Then $\text{ord}(\Pi^{S_2}) = 1$.

$$f_{\Pi^{S_1}}(S') = \bigcap_{S^* = f_{\Pi^{S_1}}(S^*)} S^* \quad (2)$$

$$= m(\Pi^{S_1}) \quad (3)$$

$$= m(\Pi_2^{S_1} \cup \Pi_1^{S'} \cup \Pi_0). \quad (4)$$

Let S^* be $S^* = m(\Pi_2^{S_1} \cup \Pi_1^{S'} \cup \Pi_0)$ and S^{**} be the set obtained by removing from $m(\Pi_2^{S_1} \cup \Pi_1^{S'} \cup \Pi_0)$ the atoms generated by the rules from $\Pi_2^{S_1} \setminus \Pi_2^{S_2}$. S^{**} sets corresponds to the S^* sets from the formula 2. Let S'' be S' without the atoms generated by $\Pi_2^{S_1} \setminus \Pi_2^{S_2}$. Then

$$S'' = m(\Pi_2^{S_2} \cup \Pi_1^{S''} \cup \Pi_0).$$

Results that $S'' \subseteq S'$ and $S'' = f_{\Pi^{S_2}}(S'')$.

We proved that for every fixpoint S' of $f_{\Pi^{S_1}}(\cdot)$, $S' = f_{\Pi^{S_1}}(S')$ there is S'' a fixpoint of $f_{\Pi^{S_2}}(\cdot)$, $S'' = f_{\Pi^{S_2}}(S'')$ such that $S'' \subseteq S'$. Consequently, $\inf\{S'' \mid S'' = f_{\Pi^{S_2}}(S'')\} \subseteq \inf\{S' \mid S' = f_{\Pi^{S_1}}(S')\}$ and $f_{\Pi}(S_1) \supseteq f_{\Pi}(S_2)$.

Proposition 3. *Let Π be a stratified LP2N with $\text{ord}(\Pi) \leq 2$. The following relations are true:*

$$\text{lfp}(f_{\Pi}^2(\cdot)) \subseteq \inf\{S \mid S = f_{\Pi}(S)\} = \bigcap_{S \in AS(\Pi)} S,$$

$$\bigcup_{S \in AS(\Pi)} S = \sup\{S \mid S = f_{\Pi}(S)\} \subseteq \text{gfp}(f_{\Pi}^2(\cdot)).$$

Consequently, $\text{lfp}(f_{\Pi}^2(\cdot))$ and $\text{gfp}(f_{\Pi}^2(\cdot))$ are an approximation of the answer set semantics for the logic program Π .

Proof: $f_{\Pi}^2(\cdot)$, is monotone and, according to Tarski theorem, it has smallest and greatest fixpoints. The fixpoints of $f_{\Pi}(\cdot)$ are also fixpoints for $f_{\Pi}^2(\cdot)$ and the theorem is proved.

Remark 6. The result indicated by proposition 3 extends the approximation provided by the well-founded semantics for the answer set semantics at LP2Ns. Well-founded semantics is a well known approximation of the answer set semantics for logic programs with only one negation as failure operator ([3]). not_1 is equivalent with classical *not* operator and the newly proposed approximation is identical with the well-founded semantics for LP2Ns Π with $\text{ord}(\Pi) \leq 1$.

Example 2. Let Π be the LP2N from example 1. To compute the approximation of the answer set semantics, the following sequence of sets of atoms will be built: $S_0 = \emptyset$, $S_1 = f_{\Pi}^2(S_0), \dots, S_i = f_{\Pi}^2(S_{i-1}), \dots$. The process will stop when first S_n is encountered such that $S_n = S_{n+1}$. $S_n = lfp(f_{\Pi}^2(\cdot))$. Let us build this sequence for the program Π . Π^{S_0} is the program $\{a \leftarrow not_1 b, b \leftarrow not_1 a, c \leftarrow\}$ and $f_{\Pi}(S_0) = \{c\}$. $\Pi^{\{c\}} = \Pi^{S_0}$ and $S_1 = f_{\Pi}^2(S_0) = \{c\}$. $S_2 = S_1$ and, consequently, $lfp(f_{\Pi}^2(\cdot)) = gfp(f_{\Pi}^2(\cdot)) = \{c\}$. For this example, the approximation of the answer set semantics is identical with the answer set semantics.

5 Properties of the approximation for the answer sets semantics

Proposition 4. *Let Π be a stratified LP2N with $ord(\Pi) \leq 2$ and a an atom, $a \in HB(\Pi)$. We build the LP2N Π' adding to Π the rule*

$$p \leftarrow not_2 a$$

where p is a new atom, $p \notin HB(\Pi)$. In these conditions

$$lfp(f_{\Pi'}^2(\cdot)) = \begin{cases} lfp(f_{\Pi}^2(\cdot)) & \text{if } a \in lfp(f_{\Pi}^2(\cdot)) \\ lfp(f_{\Pi}^2(\cdot)) \cup \{p\} & \text{if } a \notin gfp(f_{\Pi}^2(\cdot)) \end{cases}$$

Proof: Obvious, Π' is a stratified LP2N, hence $lfp(f_{\Pi'}^2(\cdot))$ exists. Let M be $M = lfp(f_{\Pi}^2(\cdot))$. Then $M = f_{\Pi}^2(M)$ and for any M_1 such that $M_1 = f_{\Pi}^2(M_1)$ then $M \subseteq M_1$. Let $\bar{M} = f_{\Pi}(M) = gfp(f_{\Pi}^2(\cdot))$. There are two cases.

– Case 1. $a \in M$. Then

$$\begin{aligned} f_{\Pi'}^2(M) &= f_{\Pi'}(f_{\Pi'}(M)) \\ &= f_{\Pi'}(\bigcap_{S'=f_{\Pi'}(M)} S') \\ &= f_{\Pi'}(\bigcap_{S'=f_{\Pi}(M)} S') \\ &= f_{\Pi'}(f_{\Pi}(M)) \\ &= f_{\Pi'}(\bar{M}). \end{aligned}$$

because $\Pi'^M = \Pi^M$. But $M \subseteq \bar{M}$, so $a \in \bar{M}$ and $f_{\Pi'}(\bar{M}) = f_{\Pi}(\bar{M}) = M$.

It results $M = f_{\Pi'}^2(M)$. Let us prove now that M is the smallest fixpoint of $f_{\Pi'}^2(\cdot)$. Suppose that there is M' such that $M' = f_{\Pi'}^2(M')$ and $M' \subset M$. $p \notin M$ so $p \notin M'$. Consequently, $a \in f_{\Pi'}(M')$ and M' is also a fixpoint of $f_{\Pi}^2(\cdot)$. But M is the smallest $f_{\Pi}^2(\cdot)$ fixpoint that contradicts with $M' \subset M$.

– Case 2. $a \notin \bar{M}$. Let M^* be $M^* = M \cup \{p\}$. Then

$$\begin{aligned} f_{\Pi'}^2(M^*) &= f_{\Pi'}(f_{\Pi'}(M^*)) \\ &= f_{\Pi'}(\bigcap_{S'=f_{\Pi'}(M^*)} S') \\ &= f_{\Pi'}(\bigcap_{S'=f_{\Pi}(M) \cup \{p\}} S') \\ &= f_{\Pi'}(\bar{M} \cup \{p\}) \\ &= f_{\Pi'}(\bar{M}) \end{aligned}$$

because $\Pi'^{M^*} = \Pi^M \cup \{p \leftarrow\}$ and $p \notin HB(\Pi)$. Because $a \notin \overline{M}$, $f_{\Pi'}(\overline{M}) = f_{\Pi \cup \{p \leftarrow\}}(\overline{M}) = f_{\Pi}(\overline{M}) \cup \{p\} = M \cup \{p\} = M^*$. Let us prove now that M^* is the smallest fixpoint of $f_{\Pi'}^2(\cdot)$. Suppose that there is M' such that $M' = f_{\Pi'}^2(M')$ and $M' \subset M^*$. $a \notin M^*$, so $a \notin M'$. Results that

$$\begin{aligned} M' &= f_{\Pi'}^2(M') \\ &= f_{\Pi'}(f_{\Pi \cup \{p \leftarrow\}}(M')) \\ &= f_{\Pi'}(f_{\Pi}(M') \cup \{p\}) \\ &= f_{\Pi'}(f_{\Pi}(M')). \end{aligned}$$

Again, there are two possibilities.

- Case 2a) $a \in f_{\Pi}(M')$. $f_{\Pi'}(f_{\Pi}(M')) = f_{\Pi}(f_{\Pi}(M'))$. M' is a fixpoint of $f_{\Pi}^2(\cdot)$ and, consequently, $M' \subseteq \overline{M}$. Results that $a \in \overline{M}$, which is a contradiction.
- Case 2b) $a \notin f_{\Pi}(M')$. $f_{\Pi'}(f_{\Pi}(M')) = f_{\Pi \cup \{p \leftarrow\}}(f_{\Pi}(M')) = f_{\Pi}(f_{\Pi}(M')) \cup \{p\}$. Results that $M' \setminus \{p\} = f_{\Pi}^2(M' \setminus \{p\})$ and $M \subseteq M' \setminus \{p\}$, which contradicts with $M' \subset M^*$.

It is natural to define that an intelligent agent with a set of premises Π knows that an atom a is true if and only if $a \in lfp(f_{\Pi'}^2(\cdot))$ regarding the approximation of the answer set semantics. An intelligent agent will not know that an atom a is true if $a \notin gfp(f_{\Pi}^2(\cdot))$. The newly introduced atom p helps to determine when $a \notin gfp(f_{\Pi}^2(\cdot))$.

Proposition 5. *Let Π be a stratified LP2N with $ord(\Pi) = 1$ and a an atom, $a \in HB(\Pi)$. We build the LP2N Π' adding to Π the rules:*

$$\begin{aligned} p &\leftarrow not_1 a \\ q &\leftarrow not_2 p \end{aligned}$$

where p, q are new atoms, $p, q \notin HB(\Pi)$. In these conditions

$$lfp(f_{\Pi'}^2(\cdot)) = \begin{cases} \bigcap_{S \in AS(\Pi)} S \cup \{q\} & \text{if } a \in \bigcup_{S \in AS(\Pi)} S \\ \bigcap_{S \in AS(\Pi)} S \cup \{p\} & \text{if } a \notin \bigcup_{S \in AS(\Pi)} S \end{cases}$$

Proof: Π' is a stratified LP2N, hence $lfp(f_{\Pi'}^2(\cdot))$ exists. Let M be $M = f_{\Pi'}^2(M)$. There are two cases.

- Case 1. $a \in \bigcup_{S \in AS(\Pi)} S$. If $p \in M$ then

$$\begin{aligned} f_{\Pi'}(M) &= \bigcap_{S = f_{\Pi'}(S)} S \\ &= \bigcap_{S = f_{\Pi \cup \{p \leftarrow not_1 a\}}(S)} S \\ &= \left(\bigcap_{\substack{S = f_{\Pi}(S) \\ a \in S}} S \right) \cap \left(\bigcap_{\substack{S = f_{\Pi \cup \{p \leftarrow\}}(S) \\ a \notin S}} S \right) \end{aligned}$$

But $ord(\Pi) = 1$ and $S = f_{\Pi}(S)$ implies $S \in AS(\Pi)$. Results that

$$f_{\Pi'}(M) = \bigcap_{S \in AS(\Pi)} S \text{ and } p \notin f_{\Pi'}(M). \quad (5)$$

If $p \notin M$ then

$$\begin{aligned} f_{\Pi'}(M) &= \bigcap_{S = f_{\Pi'}(M)(S)} S \\ &= \bigcap_{S = f_{\Pi \cup \{p \leftarrow not_1 a, q \leftarrow \}}(S)} S \\ &= \left(\bigcap_{S = f_{\Pi \cup \{p \leftarrow not_1 a\}}(S)} S \right) \cup \{q\} \end{aligned}$$

Consequently,

$$f_{\Pi'}(M) = \left(\bigcap_{S \in AS(\Pi)} S \right) \cup \{q\} \text{ and } p \notin f_{\Pi'}(M). \quad (6)$$

From 5, 6 and $M = f_{\Pi'}^2(M)$ results

$$\begin{aligned} M &= \left(\bigcap_{S \in AS(\Pi)} S \right) \cup \{q\}, \\ lfp(f_{\Pi'}^2(\cdot)) &= \left(\bigcap_{S \in AS(\Pi)} S \right) \cup \{q\} \end{aligned}$$

- Case 2. $a \notin \bigcup_{S \in AS(\Pi)} S$. The proof is similar with the case 1.

It is natural to define that an intelligent agent with a set of premises Π may believe that an atom a is true if and only if $q \in lfp(f_{\Pi'}^2(\cdot))$ in relation with the approximation of the answer set semantics. If $q \in lfp(f_{\Pi'}^2(\cdot))$ then it is guaranteed that a occurs in at least one answer set of Π .

6 Examples

Representing the Unknown. The following scenario is inspired from the scenarios described in [1] and [4]. Let us consider *alice* and *eShop* two web services where the first one is interested to make a purchase and *eShop* is a possible seller. Before *alice* buys an item from *eShop*, *alice* checks if *eShop* satisfies her criteria about what is an acceptable seller. In some cases, *alice* will request supplementary information to establish the degree of trust and safety of the seller. Below, there are listed a part of the criteria used by *alice* to establish if she can begin a transaction with another web service. The criteria are presented both as text and as logical rules (a logical rule that contains variables should be considered as a shortcut for the set of all its ground instances).

- **(alice1)** *alice* will accept to begin a transaction with a shop if the shop received high customer ratings in the past.

$$shop(X) \leftarrow highCustomerRatings(X) \quad (7)$$

- **(alice2)** *alice* will accept to begin a transaction with a shop if the shop received average customer ratings in the past and *alice* can establish that the shop is member of a trusted brand.

$$shop(X) \leftarrow averageCustomerRatings(X), trustedBrand(X) \quad (8)$$

- **(alice3)** *alice* will not begin a transaction with a shop if the shop has not high customer ratings or at least average customer ratings.

$$\neg shop(X) \leftarrow \neg highCustomerRatings(X), \neg averageCustomerRatings(X) \quad (9)$$

It should be mentioned there that the syntax of the *LP2Ns* does not allow the classical strong negation operator \neg . The classical negation \neg will be handled by using the concept of positive form [5].

- **(alice4)** If *alice* does not know if she should begin or not a transaction with a shop then she will request supplementary information from the shop about the customers support policies and about the refund policies for dissatisfied clients.

$$requestInfo(X) \leftarrow not_2 shop(X), not_2 \neg shop(X) \quad (10)$$

The affirmation *alice does not know ...* was represented as is prescribed by propositions 2 and 4.

In a business directory, *alice* has found the following information about the entity *eShop*: the customers' ratings for *eShop* are between average and high. This affirmation can be represented in logic programming by the rule:

$$highCustomerRatings(eShop) \vee averageCustomerRatings(eShop) \leftarrow$$

As the syntax of *LP2Ns* does not allow for disjunctive heads, the following rules with negation as failure will be used as a replacement (the problem of the substitutes for the disjunctive heads was discussed in [9]):

$$highCustomerRatings(eShop) \leftarrow not_1 averageCustomerRatings(eShop) \quad (11)$$

$$averageCustomerRatings(eShop) \leftarrow not_1 highCustomerRatings(eShop) \quad (12)$$

alice will use the program *II* composed from the rules (7), (8), (9), (10), (11) and (12) to establish what will be her behavior in relation with the web service *eShop*.

But

$$AS(II) = \{\{requestInfo(eShop)\}\}$$

and

$$lfp(f_{II}^2(.)) = \{requestInfo(eShop)\}.$$

Therefore, Π answers *Yes* to the query $requestInfo(eShop)$ both in answer set semantics and regarding to the approximation of the answer set semantics, which is the intended behavior. Consequently, *alice* will request from the *eShop* supplementary information about the *eShop*'s customers support and refund policies.

Representing the Closed World Assumption. Let us consider the scenario from the previous example and suppose that *alice* also includes in her interaction policies the following rule:

- (**alice5**) *alice* will accept to begin a transaction with a shop only if the shop accepts credit card payments and has fast delivery policies. If one of these requirements is not satisfied then *alice* will not begin transactions with the shop. This policy is represented by the following logic rules:

$$shop(X) \leftarrow acceptCards(X), fastDelivery(X) \quad (13)$$

$$\neg shop(X) \leftarrow \neg acceptCards(X) \quad (14)$$

$$\neg shop(X) \leftarrow \neg fastDelivery(X) \quad (15)$$

Let us suppose that *alice* already determined that *eShop* satisfies at least one of the above two conditions:

$$acceptCards(eShop) \leftarrow not_1 fastDelivery(eShop) \quad (16)$$

$$fastDelivery(eShop) \leftarrow not_1 acceptCards(eShop) \quad (17)$$

If *alice* considers that she has complete positive information about the predicates *acceptCards* and *fastDelivery* then she can use the *Closed World Assumption* for these predicates expressed by the rules:

$$\begin{aligned} \neg acceptCards(X) &\leftarrow \neg M \text{ } acceptCards(X) \\ \neg fastDelivery(X) &\leftarrow \neg M \text{ } fastDelivery(X) \end{aligned}$$

In these two rules, we used the syntax of the epistemic logic programs [6] where M is the *believe* operator. But we want to use the syntax of the *LP2Ns*, so the above two rules will be transformed as is prescribed by the propositions 1 and 5.

$$noCards(X) \leftarrow not_1 acceptCards(X) \quad (18)$$

$$believeAcceptCards(X) \leftarrow not_2 noCards(X) \quad (19)$$

$$\neg acceptCards(X) \leftarrow not_2 believeAcceptCards(X) \quad (20)$$

$$noFastDelivery(X) \leftarrow not_1 fastDelivery(X) \quad (21)$$

$$believeFastDelivery(X) \leftarrow not_2 noFastDelivery(X) \quad (22)$$

$$\neg fastDelivery(X) \leftarrow not_2 believeFastDelivery(X) \quad (23)$$

Let Π be the $LP2N$ composed from the rules (13), (14), (15), (16), (17), (18), (19), (20), (21), (22) and (23). Then

$$AS(\Pi) = \{\{believeAcceptCards(eShop), believeFastDelivery(eShop)\}\}$$

and

$$lfp(f_{\Pi}^2(.)) = \{believeAcceptCards(eShop), believeFastDelivery(eShop)\}.$$

Consequently, regarding to Π , *alice* cannot establish for sure if she will begin or not a transaction with *eShop*. To illustrate the gain of accuracy provided by $LP2Ns$, let us consider the previous information described by an extended logic program [2]. Let Π' be the extended logic program composed by the following rules:

$$\begin{aligned} shop(X) &\leftarrow acceptCards(X), fastDelivery(X) \\ \neg shop(X) &\leftarrow \neg acceptCards(X) \\ \neg shop(X) &\leftarrow \neg fastDelivery(X) \\ acceptCards(eShop) &\leftarrow not\ fastDelivery(eShop) \\ fastDelivery(eShop) &\leftarrow not\ acceptCards(eShop) \\ \neg acceptCards(X) &\leftarrow not\ acceptCards(X) \\ \neg fastDelivery(X) &\leftarrow not\ fastDelivery(X) \end{aligned}$$

Regarding to the answer set semantics for extended logic programs, the Π' answer to the question $\neg shop(eShop)$ is *Yes*. This answer is somehow improper because *alice*'s information does not exclude the possibility that *eShop* both to accept credit cards and to have fast delivery policies.

If a Web service includes interaction rules represented by $LP2Ns$ then it should also have access to a logic programming reasoning system. The Web services knowledge engineers will not represent concepts like *know*, *not know*, *believe* and *not believe* at low level by using directly the not_2 operator. Instead, they will use the K and M operators as is the case for the epistemic logic programs and these operators will be automatically replaced by the logic programming reasoning system with groups of rules based on not_2 . The replacement will be done in accordance with the propositions 4 and 5. The logic programming reasoning system will compute the approximation of the answer set semantics over the transformed program Π . This operation is accomplished by starting with the empty set of atoms and applying repeatedly $f_{\Pi}^2(.)$ to the current set of atoms until a fix point is reached. This fix point is $lfp(f_{\Pi}^2(.))$. $gfp(f_{\Pi}^2(.))$ is computed using the formula $gfp(f_{\Pi}^2(.)) = f(lfp(f_{\Pi}^2(.)))$.

7 Conclusion

The paper considers the problem of establishing interaction compatibility between web services whose interface behaviors are specified in terms of reactive rules. Different types of models from logic programming were already used to give answers at that question but sometimes are needed models that are able

to represent what an intelligent agent knows or believes. Epistemic logic programs are a good model for this job but, computationally, they are too complex. A new kind of logic programs with two types of negation as failure operators is proposed to support the reasoning needs of a web service. The answer set semantics for these programs and its polynomial approximation are able to represent epistemic information. The obtained mathematical results indicate how to incorporate epistemic into $LP2N$ and how transform epistemic logic programs into equivalent $LP2Ns$. The usability of the $LP2Ns$ is illustrated by two knowledge representation examples where a web service tries to establish its interaction compatibility with another web service by performing a logic reasoning to determine if its interaction compatibility criteria are satisfied.

Acknowledgments. The author acknowledges anonymous reviewers for their valuable comments and suggestions. The research was supported by the Romanian National University Research Council (AT grant 27GR/11.05.2007, Code 102).

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Torroni, P.: Policy-based reasoning for smart web service interaction. International Workshop Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006). In conjunction with ICLP2006, part of FLOC2006, Seattle, Washington, (2006).
2. Baral, C., Gelfond, M.: Logic Programming and Knowledge Representation. Journal of Logic Programming, 19,20:73-148 (1994).
3. Baral, C., Subrahmanian, V.: Duality between alternative semantics of logic programs and nonmonotonic formalisms. In International Workshop on Logic Programming and Nonmonotonic Reasoning, A. Nerode, W. Marek, and V. Subrahmanian, Eds. MIT Press, Washington DC., (1991) 69-86.
4. Bry, F., Eckert, M.: Twelve Theses on Reactive Rules for the Web. Proceedings of Workshop Reactivity on the Web, Munich, Germany, 2006.
5. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing, 9:365-385 (1991).
6. Gelfond, M.: Logic Programming and Reasoning with Incomplete Information. Annals of Mathematics and Artificial Intelligence, 12:89-116 (1994).
7. Lloyd, J. W.: Foundations of Logic Programming. Springer-Verlag, second edition, (1987).
8. Mascardi, V., Casella, G.: Intelligent Agents that Reason about Web Services: a Logic Programming Approach. International Workshop Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006). In conjunction with ICLP2006, part of FLOC2006, Seattle, Washington, August 10-22, (2006).
9. Schaerf, M.: Negation and minimality in disjunctive databases. Journal of Logic Programming, (23):63-87, (1995).
10. Zhang, Y.: Epistemic Reasoning in Logic Programs, IJCAI-07, (2007) 647-652.