Supporting Open and Closed World Reasoning on the Web*

Carlos Viegas Damásio[1], Anastasia Analyti[2], Grigoris Antoniou[2,3], and Gerd Wagner[4]

[1] Centro de Inteligência Artificial, Universidade Nova de Lisboa, Caparica, Portugal
[2] Institute of Computer Science, FORTH-ICS, Greece
[3] Department of Computer Science, University of Crete, Greece
[4] Institute of Informatics, Brandenburg Univ. of Technology at Cottbus Germany
analyti@ics.forth.gr, antoniou@ics.forth.gr, cd@di.fct.unl.pt, G.Wagner@tu-cottbus.de

Abstract. In this paper general mechanisms and syntactic restrictions are explored in order to specify and merge rule bases in the Semantic Web. Rule bases are expressed by extended logic programs having two forms of negation, namely strong (or explicit) and weak (also known as default negation or *negation-as-failure*). The proposed mechanisms are defined by very simple modular program transformations, and integrate both open and closed world reasoning. These program transformations are shown to be appropriate for the two major semantics for extended logic programs: answer set semantics and well-founded semantics with explicit negation. Moreover, the results obtained by both semantics are compared.

1 Introduction

The Semantic Web [3] aims at defining formal languages, and corresponding tools, enabling automated processing and reasoning over (meta-)data available from the Web. Logic and knowledge representation play a central role, but the distributed and world-wide nature of the Web bring new interesting research problems. In particular, the widely recognized need of having rules in the Semantic Web [13, 17] has restarted the discussion of the fundamentals of closed-world reasoning and the appropriate mechanisms to implement it in rule systems, such as the computational concept of *negation-as-failure*.

The classification if a predicate is completely represented or not is up to the owner of the knowledge base: the owner must know for which predicates there is complete information and for which there is not. Unfortunately, neither classical logic nor standard Prolog supports the distinction between "closed" and "open" predicates. Classical logic supports only open-world reasoning. On the contrary,

* This research has been partially funded by European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net)

most Prolog systems support only closed-world reasoning, as *negation-as-failure* is the only negation mechanism supported (a notable exception is XSB [18]). We resort to two major semantics of extended logic programs, namely *answer set semantics* [10], and *well-founded semantics with explicit negation* [15, 1], which have two forms of negation: weak and strong. Weak negation is an appropriate rendering of the mechanism of nonmonotonic negation-as-failure, and strong negation allows the user to express negative knowledge and is monotonic. The combination of these two forms of negation allow the distinction between open and closed predicates, as will be illustrated by their application to the declaration and construction of rule bases in the Semantic Web.

The paper is organized as follows. In Section 2, the use of extended logic programming is explored to represent open and closed world reasoning, providing general mechanisms for achieving this. Section 3 defines new language mechanisms for sharing and integrating knowledge in the Semantics Web. In Section 4, the transformational semantics is provided for the constructs presented. The paper finishes with comparisons and conclusions.

2 Open and Closed World Assumption

Rule bases are sets of extended logic programming rules of the form

$$L_0 \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots \sim L_n \tag{1}$$

where each L_i (with $0 \leq i \leq n$) is an objective literal, i.e. either an atom $A(\bar{t})$ or the strong negation of an atom $\neg A(\bar{t})$, where $\bar{t}$ is a sequence of terms. Variables are prefixed with a question mark symbol (`?`), therefore names for predicates, constants and function symbols can start with small and capital letters. It is assumed that a fixed first order logic alphabet is given, and only extended Herbrand interpretations are considered (sets of objective literals). In particular, a non-ground rule in an extended logic program stands for the set of ground rules obtained by instantiating logical variables with elements from the Herbrand universe. Notice that implicitly we are using a *domain closure assumption* which might not be acceptable in some situations. Without loss of generality, only ground programs are considered in the subsequent theoretical results. Furthermore, we restrict the discussion to DATALOG programs over a finite number of constants in order to guarantee decidability of reasoning. We define by $\mathcal{C}_{SEM}(P)$ the set of *objective literals* which are obtained from the extended logic program P under semantics SEM, where $SEM = WFSX$ or $SEM = AS$. Here we consider only sceptical answer set semantics [10], denoted by subscript $SEM = AS$, and well-founded semantics with explicit negation [15, 1], denoted by subscript $SEM = WFSX$. For inconsistent programs, both these semantics adopt an explosive approach by letting $\mathcal{C}_{SEM}(P)$ be the set of all objective literals. The reader is referred to the literature for details.

Example 1. Consider the following program expressing immigration laws of an imaginary country. Notice that all the rules are objective, i.e. do not use weak negation.

```
Enter(?p) ← CountryEU(?c), citizenOf(?p,?c).
Enter(?p) ← ¬ CountryEU(?c), citizenOf(?p,?c), ¬ RequiresVisa(?c).
Enter(?p) ← ¬ CountryEU(?c), citizenOf(?p,?c),
            RequiresVisa(?c), HasVisa(?p).
```

Predicate `Enter/1` captures the following laws:

- A citizen of European Union can enter the country.
- A non European Union citizen can enter the country if a visa is not required.
- A non European Union citizen can enter the country if a visa is required and he/she has it.

These rules are complemented with the following knowledge, where it is assumed that the list of European Union countries is exhaustive:

```
CountryEU(Austria).
⋮

¬ CountryEU(China).
¬ CountryEU(Djibuti).

¬ RequiresVisa(Bulgaria).
¬ RequiresVisa(?c) ← CountryEU(?c).
RequiresVisa(China).
```

Some facts about Anne, Boris, Chen and Dil finish the program:

```
citizenOf(Anne,Austria).
citizenOf(Boris,Bulgaria).
citizenOf(Chen,China).          HasVisa(Chen).
citizenOf(Dil,Djibuti).         HasVisa(Dil).
```

The arbitrary uncontrolled use of weak negation in the Semantic Web is regarded problematic and unsafe. However, local closed world assumptions and *scoped negation-as-failure* have been identified as desirable and necessary for the Semantic Web [11, 14, 16, 23, 2]. The difficulty lies on the definition of simple mechanisms that can be easily explained to ordinary users, and have nice mathematical properties. For this reason, we propose a classification of predicates which cover the whole gamut of alternatives. The classes of *objective*, *open* and *closed* predicates impose some restrictions on the use of weak negation in the rules defining a predicate A in the Semantic Web, which are summarized in Figure 1. The top-half boxes contain the user's predicate definitions and are always sets of objective rules, i.e. rules which do not contain weak negation but might contain strongly negated literals, in particular the head of rules might be $A(\bar{t})$ or $\neg A(\bar{t})$. The bottom-half boxes contain special rules, added by the system, which characterize each type of predicate. Additionally, it is required that objective, open and closed predicates do not use (directly or indirectly) unrestricted predicates on their definitions. This prevents unintended use of weak negation in the Semantic Web. The unrestricted predicates are designated *normal* (or ordinary) predicates, adopting the usual logic programming accepted terminology.

Thus, objective predicates are defined by rules which do not contain weak negation at all. Since strong negation is monotonic, then these predicates can

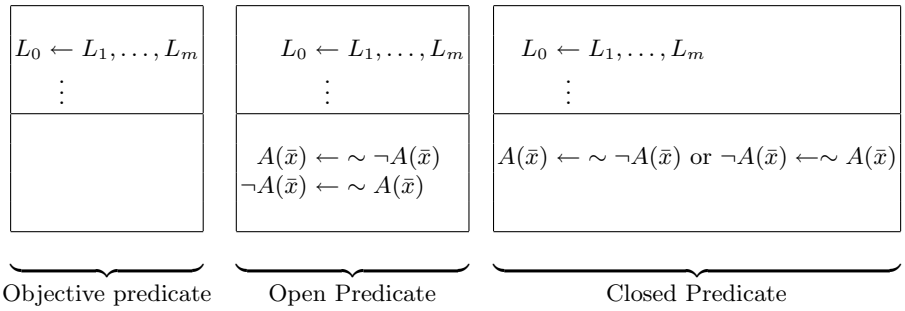

Fig. 1. Declarations for a predicate A (the predicate of L_0 is A)

be freely used in the Semantic Web without any restriction. These predicates are *partial* since it may be the case that neither $A(\overline{c})$ nor $\neg A(\overline{c})$ hold in a model (see [12, 2] for more details), where $\overline{c}$ is a sequence of constants. On the other hand, open predicates have the following two additional rules, denoted by $openRules(A)$:

$$A(\overline{x}) \leftarrow\sim \neg A(\overline{x}) \qquad \neg A(\overline{x}) \leftarrow\sim A(\overline{x})$$

In answer set semantics, these specify that either $A(\overline{c})$ is true or $\neg A(\overline{c})$ is true in each model (answer set), thus forcing *totalness*.

Finally, closed predicates are complemented by one and only one of the previous two rules, called *default closure rules*, and denoted by $negClosure(A)$ and $posClosure(A)$, respectively. This provides a mechanism for making closed world assumptions: either by making true what is not concluded false or by making false what is not concluded true.

Example 2. Returning to Example 1, start by assuming that all predicates are objective. The following conclusions are obtained from the original program, both with AS and $WFSX$ semantics:

```
Enter(Anne)   Enter(Chen)
```

Interestingly, `Enter(Boris)` is not concluded because it is not known that Bulgaria is a European Union country and also it is not known that it is not a European Union country! One way to circumvent this situation is to state that predicate `CountryEU/1` is open. Notice that it does not make sense to state that `CountryEU/1` is closed since EU is evolving and new countries in the near future might integrate EU, namely Bulgaria[5]. By declaring `CountryEU/1` open, the following two rules are added:

```
   CountryEU(?c) ← ∼ ¬ CountryEU(?c)
¬ CountryEU(?c) ← ∼ CountryEU(?c)
```

[5] This is a simple-minded solution to the problem of knowledge update.

From the new program and using AS semantics, it is concluded that Boris can enter the country. The argument is the following: if Bulgaria is a member of EU, then by the first rule Boris can enter the country; if Bulgaria is not a European Union country, since a Visa is not required for Bulgaria, then Boris can also enter the country. WFSX semantics is not capable of doing this case analysis and therefore this conclusion is not obtained.

Finally, consider the situation where `Enter/1` and ¬`RequiresVisa/1` are exhaustive. These predicates can be closed, by introducing the following rules:

$$\neg \texttt{Enter(?p)} \leftarrow \sim \texttt{Enter(?p)}$$
$$\texttt{RequiresVisa(?c)} \leftarrow \sim \neg \texttt{RequiresVisa(?c)}$$

The first rule expresses that if by the immigration laws cannot be concluded that a person can enter the country, then that person cannot enter the country. The second rule states that the list of countries, for which it is not requested a Visa, is closed. This means that it is requested a Visa for the non-listed countries. Both under WFSX and AS semantics, it is now concluded that Dil can enter the country.

Notice that in the move from all predicates being objective to some being open and then closed, new conclusions might be obtained, as the following major Theorem shows:

Theorem 1. *Let A be an objective predicate in extended logic program P where all predicates are either objective or open. Then,*

- $\mathcal{C}_{SEM}(P) \subseteq \mathcal{C}_{SEM}(P \cup openRules(A))$
- $\mathcal{C}_{SEM}(P \cup openRules(A)) \subseteq \mathcal{C}_{SEM}(P \cup posClosure(A))$
- $\mathcal{C}_{SEM}(P \cup openRules(A)) \subseteq \mathcal{C}_{SEM}(P \cup negClosure(A))$

with $SEM = AS$ or $SEM = WFSX$.

For the case of $WFSX$ semantics the first containment is in fact an equality, i.e. $\mathcal{C}_{WFSX}(P) = \mathcal{C}_{WFSX}(P \cup openRules(A))$. The previous theorem cannot be generalized when some predicate is closed in P. This is expected due to the non-monotonic nature of weak negation under both AS semantics and WFSX semantics.

Example 3. Consider the original program of Example 1 but now `Enter/1` is declared closed with the rule:

$$\neg \texttt{Enter(?p)} \leftarrow \sim \texttt{Enter(?p)}$$

It can be concluded with WFSX and AS semantics that ¬ `Enter(Boris)` and ¬ `Enter(Dil)`. Now by declaring `CountryEU/1` open, ¬ `Enter(Dil)` is not concluded anymore with WFSX and AS. As previously, `Enter(Boris)` is concluded with AS but not with WFSX.

Notice that under WFSX no new objective conclusions are obtained by declaring predicates open. This is expected since entailment in WFSX can be computed in polynomial time, while entailment in AS is coNP-complete. This is the tradeoff between expressivity and complexity of reasoning. However, WFSX and AS semantics are not unrelated:

Theorem 2. *[15] Let P be an extended logic program, then $\mathcal{C}_{WFSX}(P) \subseteq \mathcal{C}_{AS}(P)$.*

WFSX is a tractable semantics which approximates AS semantics, and therefore is a good candidate for defining the semantics of rule bases in the Semantic Web. However, the existence of an undefined truth-value in WFSX might affect the intuition in some particular cases, namely for closed predicates; this is the price to pay for guaranteeing tractability of reasoning. Aside that, both semantics assure the monotonicity of reasoning in the presence of only objective and open predicates:

Theorem 3. *Let P and Q be two extended logic programs where all predicates are either objective or open. Then,*

- $\mathcal{C}_{AS}(P) \subseteq \mathcal{C}_{AS}(P \cup Q)$
- $\mathcal{C}_{WFSX}(P) \subseteq \mathcal{C}_{WFSX}(P \cup Q)$

Obviously, the previous result does not hold whenever closed predicates are included in P or Q. The above theorems are explored in the next section for defining modular programming techniques to be used in the Semantic Web.

3 Modularity in the Semantic Web

In this section we study the mechanisms in order to be able to express the necessary context to use strong and weak negations safely in the Semantic Web environment. The discussion is abstract and independent of any rule engine. Currently, there is no notion of scope or context in the Semantic Web: all knowledge is global and all kinds of unexpected interactions can occur. The success of the Semantic Web is impossible without any form of modularity, encapsulation, information hiding and access control. The issue of modularity in logic programming has been actively investigated during the 90s, for a survey see [4]. Here we follow a typical approach similar to the import/export mechanisms of Prolog, but we will be concerned with the combination of open and closed world reasoning and other particularities of the Semantic Web. In particular, the following four levels of context and their interaction must be taken into account:

- The Semantic Web context;
- The application context, corresponding to the context where a user or Semantic Web agent loads, asserts or consumes the knowledge provided by rule bases in the Semantic Web;
- The rule base context, where the Semantic Web developer encapsulates a set of related rules and facts (predicates);
- The predicate context, which can be either global or local;

Rule bases are made available in the Semantic Web, and users or applications load or assert them explicitly into their application contexts. The connection to an external knowledge base should always be equivalent to loading it locally,

DefinesDecl ::=
  [*RuleBaseIRI*] `"defines"` [*ScopeDecl*] *PredList* [`"visible to"` *RuleBaseList*] `"."`
UsesDecl ::= [*RuleBaseIRI*] `"uses"` *PredList* [`"from"` *RuleBaseList*] `"."`

ScopeDecl ::= `"global"` | `"local"` | `"internal"`
PredList ::= *PredicateDecl* (`","` *PredicateDecl*)*
RuleBaseList ::= *RuleBaseIRI* (`","` *RuleBaseIRI*)*
PredicateDecl ::= [`"objective"` | `"open"` | `"closed"` [`"¬"`] | `"normal"`] *PredicateInd*
PredicateInd ::= *AbsoluteIRI* [`"/"` Arity]
Arity ::= *Natural*
RuleBaseIRI ::= *AbsoluteIRI*

Fig. 2. The `defines` and `uses` declarations

but without the need to explicitly do that. When a user or application loads or asserts knowledge, it may express that nonmonotonic reasoning forms may be rejected or allowed, or can force the deduction mechanisms to use only rules which extract safe knowledge in the Semantic Web context. The knowledge base programmer may use nonmonotonic constructs, knowing that these constructs might be inhibited or forbidden. The producer of knowledge might also express that the predicates he/she is declaring cannot be defined elsewhere, and may declare hidden predicates which are not visible in the Semantic Web. Furthermore, a knowledge base might use all the available knowledge in the application context, or get it explicitly from particularly loaded rule bases. By default, reasoning in the Semantic Web must be monotonic.

The challenge is to provide simple mechanisms in order to guarantee the fulfilment of the previous requirements. Obviously, the syntax of extended logic programs should be augmented with declarations to state the visibility of a predicate, its context, and whether it is normal (i.e. unrestricted), objective, open or closed. It is also necessary to express how external information to the knowledge base is incorporated into it. These can be attained with the declarations `defines` and `uses` with the syntax in BNF notation presented in Figure 2. The `defines` declaration specifies which predicates are defined (and exported) in the knowledge base, their scope and visibility, as well as type. The `uses` declaration describes which predicates are used (imported) from other rule bases or from the Semantic Web, and might change the original type of the predicate. Notice that predicates and rule bases are all identified by absolute IRIs (Internationalized Resource Identifiers [7]). When a predicate A is declared `closed` (resp. `closed` ¬) then the $posClosure(A)$ (resp. $negClosure(A)$) rule is implicitly added to the program. If the predicate is declared `open`, then both rules are added, as described in the previous section.

The scope plays a fundamental part, and describes what is the context of the predicate(s) and may take one of the following values, with the following corresponding limitations and meaning:

`"global":` a predicate declared global is visible outside the knowledge base, and intends to capture predicates being defined in the Semantic Web. Moreover,

the predicate can be defined elsewhere in other rule bases but it must be either objective or open[6]. Additionally, it can be optionally declared which rule bases can use the predicate; if omitted, it can be used everywhere.

`"local":` a local predicate can be used outside the rule base where it has been defined, but cannot be defined by any other knowledge base in the Semantic Web. A local predicate can be of any type (objective, open, closed and normal) and, as before, the user can state the rule bases where it can be used.

The rule base defines the scope for a closed predicate, and the closure rule may be inhibited by the consumer of the knowledge in the `uses` statement. If the predicate is normal, any form of negation can be used in its definition, and its use can be forbidden by the consumer of the knowledge, again with the `uses` statement.

`"internal":` predicate is internal to the rule base and cannot be used outside the rule base. Again, the rule base defines the scope for the evaluation of weak negation.

By default a predicate is global and open, and visible to any rule base in the Semantic Web. Also, all predicates in the RDF and RDFS vocabularies are global and open. Thus, the user doesn't have to state explicitly the scope and type of predicates in all rule bases. Furthermore, this guarantees monotonicity of reasoning. It is not practically possible to guarantee that a local predicate is not redefined multiple times in the Semantic Web. However, any implementation will not allow loading knowledge bases which define a local or global predicate defined local in another loaded rule base.

The visibility provides a basic security mechanism, but trust and authorization could be much improved, for instance using the PeerTrust language [9]. These issues are orthogonal to present proposal but can be easily integrated due to the logical nature of our work. The `uses` declaration specifies the rule bases providing the definitions of global and local predicates that can be used by the importing rule base. The scope of the imported predicated is given by a corresponding `defines` statement in the rule base, whenever it exists. If the `from` list in the `uses` declaration (Fig. 2) is omitted then these predicates can be imported from any available knowledge base. Notice that the importer can specify what types of predicates (reasoning) he/she is willing to accept, and the default type is open. The exporter must provide the answers according to the cases specified in Table 1.

For instance, suppose that a rule base $< RB_A >$ `defines` a closed predicate P with: $< RB_A >$ `defines local closed` P.

However, the `uses` statement in rule base $< RB_B >$ declares that it is only willing to accept the conclusions obtained by opening the predicate P in $< RB_A >$: $< RB_B >$ `uses open` P `from` $< RB_A >$.

Rule base $< RB_A >$ should only provide answers to queries of P from $< RB_B >$ as if all closed predicates in $< RB_A >$ were open. If $< RB_B >$

[6] For simplicity, this constraint is not enforced in the grammar.

Table 1. Combination of reasoning modes

uses (importer) \ defines (exporter)	objective	open	closed	normal
normal	objective	open	closed	normal
closed	objective	open	closed	error
open	objective	open	open	error
objective	objective	objective	objective	error
	objective	open	closed	normal

`uses` predicate P of $< RB_A >$ in objective mode, then all predicates in rule base $< RB_A >$ are considered objective when computing the queries to P from $< RB_B >$. In other words, the reasoning mode should also be propagated to the predicates used in $< RB_A >$, whenever these predicates are necessary to answer the original query. Finally, we would like to note that there are subtle issues involved in the above mechanisms, namely the possibility of mutual dependencies between rule bases, which should be addressed in implementations. A runtime error is thrown when the exporter declares a local predicate normal but the importer uses one of the limited predicate reasoning forms: objective, open or closed. This behaviour corresponds to rejecting by the importer the uncontrolled use of weak negation in the Semantic Web. Note again that according to the results of the previous section, the default declarations guarantee that reasoning is monotonic.

A knowledge base might define and use the same predicate, but not all combinations are possible. The various allowed combinations are presented in Table 2.

Table 2. Defining and using the same predicate

defines \ uses	global	local	internal
global	allowed	error	error
local	error	error	error
internal	allowed	allowed	error

Obviously, it is an error to globally or locally define a used local predicate; this goes against the notion that there is a sole provider for a local predicate. However, it is allowed to internally redefine a local predicate of a different rule base, since it is not made public. In particular, one might close an objective local predicate of a different provider since this is only for internal use.

The several combinations are illustrated with the next example.

Example 4. Consider the knowledge bases in the Semantic Web identified by IRIs `<http://www.eu.int>`, `<http://gov.country>` and `<http://security.int>`. We use the namespace prefixes `eu`, `gov` and `sec` to simplify writing of IRIs in the code of Figure 3.

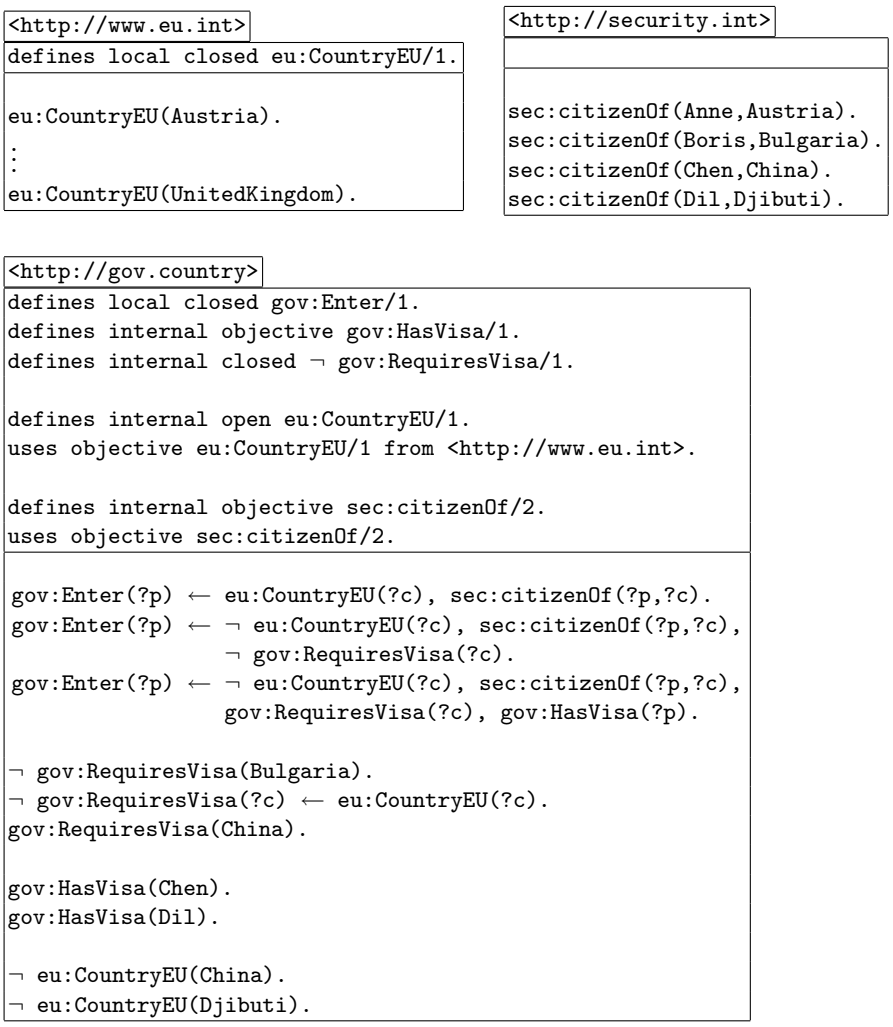

Fig. 3. Sharing of Knowledge in the Semantic Web

The simpler rule base, identified by `<http://www.eu.int>`, defines the list of European Union countries, and this list is closed. Notice that this is a proper logical definition of the CWM [5] construct `log:definitiveDocument`. The second rule base, `<http://security.int>`, provides citizenship of people, and could be

implemented in a ordinary relational database. Since no `defines` declaration is present, predicate `sec:citizenOf/2` is a global and open predicate.

The third rule base defines the immigration policies of country `<http://gov.country>`, supported by the knowledge of the other two rule bases. The first three `defines` statements are according to the discussion in Example 1; it should be noticed the mechanism for closing negative instances in `gov:RequiresVisa/1` with ¬. The country is not willing to accept the local closure of `eu:CountryEU/1` performed in `<http://www.eu.int>`. Therefore, it `uses` the predicate forcing objective mode and, in this example, only facts are requested to `<http://www.eu.int>`. Furthermore, `eu:CountryEU` is made open, for use in this rule base; this can be done since the predicate is defined to be internal. Complementary additional facts to predicate `eu:CountryEU/1` are stated in the rule base. Predicate `sec:citizenOf` is used from any providers in the Semantics Web, but it is made objective for internal use only.

The code of the figure is unsatisfactory from a security point of view. In an additional rule base, it could be added a fact stating that, for instance, `sec:citizenOf(Chen,France)`. Since rule base `<http://gov.country>` is carelessly using `sec:citizenOf/2` from the Semantic Web it imports any existing available knowledge independently of the providing rule base. This can be corrected with the statement:

```
uses objective sec:citizenOf/2 from <http://security.int>.
```

If more sources are trusted, these can be added to the `from` list. Also, `<http://security.int>` is providing confidential information to any requester. This can also be improved by specifying the authorized consumers of this knowledge base in the `visible to` list, e.g.:

```
defines global open sec:citizenOf/2 visible to <http://gov.country>.
```

4 Transformational Semantics

In this section, we define a modular program transformation capturing the semantics of each of the proposed constructs described in the previous section. For capturing the intended semantics, a single extended logic program is constructed. In order to control visibility and scope of the predicates, predicate names are transformed into a pair containing the rule base IRI and the predicate IRI[7]. In our transformation, a rule will be translated into four rules, one for each possible reasoning mode: definite (objective), open, closed and normal. This permits a modular way of independently composing the several rule bases, i.e. adding the transformational rules corresponding to a rule base does not require changing the form of the transformational rules of already handled rule bases. Suppose that a rule base r contains the rule:

$$L_0 \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n. \tag{2}$$

[7] In order to avoid name clashes it is assumed that IRIs always appear between delimiters '<' and '>'.

Accordingly, the rule is translated into the following four rules:

$$r{:}d\_L_0 \leftarrow r{:}d\_L_1, \ldots, r{:}d\_L_m, \sim r{:}d\_L_{m+1}, \ldots, \sim r{:}d\_L_n.$$
$$r{:}o\_L_0 \leftarrow r{:}o\_L_1, \ldots, r{:}o\_L_m, \sim r{:}o\_L_{m+1}, \ldots, \sim r{:}o\_L_n.$$
$$r{:}c\_L_0 \leftarrow r{:}c\_L_1, \ldots, r{:}c\_L_m, \sim r{:}c\_L_{m+1}, \ldots, \sim r{:}c\_L_n.$$
$$r{:}n\_L_0 \leftarrow r{:}n\_L_1, \ldots, r{:}n\_L_m, \sim r{:}n\_L_{m+1}, \ldots, \sim r{:}n\_L_n.$$

where if $A(\bar{t})$ is an atom of the original rule base r, the literal $r{:}x\_\neg A(\bar{t})$ in the translated rules is replaced by $\neg r{:}x\_A(\bar{t})$ (for $x \in \{d, o, c, n\}$). The prefixes d, o, c and n are used to distinguish the reasoning mode for the rule, that is, definite, open, closed, and normal, respectively. The meaning of a predicate A in a rule base r is always given by the instances of $r{:}n\_A(\bar{c})$ and $\neg r{:}n\_A(\bar{c})$ which are true in all intended model(s), under one of the adopted semantics $WFSX$ or AS. Recall that $WFSX$ is always an approximation of AS semantics, obtaining less conclusions.

Due to space limitations, our transformational semantics ignores errors, which should be syntactically treated a priori. For example, in the case that a predicate A in rule base r is defined as objective, open, or closed then for every rule (2) with $L_0 = A(\bar{t})$ or $L_0 = \neg A(\bar{t})$, it should hold $n = m$. All the syntactical restrictions are discussed in Section 3.

The `defines` declaration is translated according to the following. First, for global and local predicates the following rules are introduced. Notice that by declaring a predicate global or local, the rule base component of the name is removed and this makes the predicate accessible to the outside world.

$$d\_A(\overline{x}) \leftarrow r{:}d\_A(\overline{x}). \qquad \neg d\_A(\overline{x}) \leftarrow \neg r{:}d\_A(\overline{x}).$$
$$o\_A(\overline{x}) \leftarrow r{:}o\_A(\overline{x}). \qquad \neg o\_A(\overline{x}) \leftarrow \neg r{:}o\_A(\overline{x}).$$
$$c\_A(\overline{x}) \leftarrow r{:}c\_A(\overline{x}). \qquad \neg c\_A(\overline{x}) \leftarrow \neg r{:}c\_A(\overline{x}).$$
$$n\_A(\overline{x}) \leftarrow r{:}n\_A(\overline{x}). \qquad \neg n\_A(\overline{x}) \leftarrow \neg r{:}n\_A(\overline{x}).$$

If predicate A is declared `open` in rule base r, the following rules are added (see column "open" of Table 1):

$$r{:}o\_A(\overline{x}) \leftarrow\sim \neg r{:}o\_A(\overline{x}). \qquad \neg r{:}o\_A(\overline{x}) \leftarrow\sim r{:}o\_A(\overline{x}).$$
$$r{:}c\_A(\overline{x}) \leftarrow\sim \neg r{:}c\_A(\overline{x}). \qquad \neg r{:}c\_A(\overline{x}) \leftarrow\sim r{:}c\_A(\overline{x}).$$
$$r{:}n\_A(\overline{x}) \leftarrow\sim \neg r{:}n\_A(\overline{x}). \qquad \neg r{:}n\_A(\overline{x}) \leftarrow\sim r{:}n\_A(\overline{x}).$$

Compare with the case when A is declared to be `closed` $\neg$ (negatively closed) or `closed` (positively closed) in rule base r (see column "closed" of Table 1):

$$r{:}o\_A(\overline{x}) \leftarrow\sim \neg r{:}o\_A(\overline{x}). \quad \textit{and} \quad \neg r{:}o\_A(\overline{x}) \leftarrow\sim r{:}o\_A(\overline{x}).$$
$$r{:}c\_A(\overline{x}) \leftarrow\sim \neg r{:}c\_A(\overline{x}). \quad \textit{or} \quad \neg r{:}c\_A(\overline{x}) \leftarrow\sim r{:}c\_A(\overline{x}).$$
$$r{:}n\_A(\overline{x}) \leftarrow\sim \neg r{:}n\_A(\overline{x}). \quad \textit{or} \quad \neg r{:}n\_A(\overline{x}) \leftarrow\sim r{:}n\_A(\overline{x}).$$

The rules in the first line make the predicate open, which corresponds to the case where the importing rule base forces open reasoning mode (see row "open" of Table 1). When the predicate is declared `objective` or `normal`, no additional rules are required.

The `uses` declaration is easier to treat, generating rules that also respect Table 1.

r `uses objective` A `from` s declaration:

$$
\begin{array}{ll}
r{:}d\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}d\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}). \\
r{:}o\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}o\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}). \\
r{:}c\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}c\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}). \\
r{:}n\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}n\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}).
\end{array}
$$

r `uses open` A `from` s declaration:

$$
\begin{array}{ll}
r{:}d\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}d\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}). \\
r{:}o\_A(\overline{x}) \leftarrow s{:}o\_A(\overline{x}). & \neg r{:}o\_A(\overline{x}) \leftarrow \neg s{:}o\_A(\overline{x}). \\
r{:}c\_A(\overline{x}) \leftarrow s{:}o\_A(\overline{x}). & \neg r{:}c\_A(\overline{x}) \leftarrow \neg s{:}o\_A(\overline{x}). \\
r{:}n\_A(\overline{x}) \leftarrow s{:}o\_A(\overline{x}). & \neg r{:}n\_A(\overline{x}) \leftarrow \neg s{:}o\_A(\overline{x}).
\end{array}
$$

r `uses closed` A `from` s declaration:

$$
\begin{array}{ll}
r{:}d\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}d\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}). \\
r{:}o\_A(\overline{x}) \leftarrow s{:}o\_A(\overline{x}). & \neg r{:}o\_A(\overline{x}) \leftarrow \neg s{:}o\_A(\overline{x}). \\
r{:}c\_A(\overline{x}) \leftarrow s{:}c\_A(\overline{x}). & \neg r{:}c\_A(\overline{x}) \leftarrow \neg s{:}c\_A(\overline{x}). \\
r{:}n\_A(\overline{x}) \leftarrow s{:}c\_A(\overline{x}). & \neg r{:}n\_A(\overline{x}) \leftarrow \neg s{:}c\_A(\overline{x}).
\end{array}
$$

r `uses normal` A `from` s declaration:

$$
\begin{array}{ll}
r{:}d\_A(\overline{x}) \leftarrow s{:}d\_A(\overline{x}). & \neg r{:}d\_A(\overline{x}) \leftarrow \neg s{:}d\_A(\overline{x}). \\
r{:}o\_A(\overline{x}) \leftarrow s{:}o\_A(\overline{x}). & \neg r{:}o\_A(\overline{x}) \leftarrow \neg s{:}o\_A(\overline{x}). \\
r{:}c\_A(\overline{x}) \leftarrow s{:}c\_A(\overline{x}). & \neg r{:}c\_A(\overline{x}) \leftarrow \neg s{:}c\_A(\overline{x}). \\
r{:}n\_A(\overline{x}) \leftarrow s{:}n\_A(\overline{x}). & \neg r{:}n\_A(\overline{x}) \leftarrow \neg s{:}n\_A(\overline{x}).
\end{array}
$$

If the importing rule base list is absent from the `uses` declaration, then instead of $s{:}d\_A(\overline{x})$, $s{:}o\_A(\overline{x})$, $s{:}c\_A(\overline{x})$ and $s{:}n\_A(\overline{x})$ in the body of the previous rules, it should be used instead, respectively, $d\_A(\overline{x})$, $o\_A(\overline{x})$, $c\_A(\overline{x})$ and $n\_A(\overline{x})$. The effect is to import all the existing knowledge regarding the predicate and which is publicly available from the several rule bases (due to space limitations, here we ignore visibility issues).

The major issue remaining to be discussed is the scope of the weak negation operator. For simplicity of discussion, it is assumed that the variables of the transformational rules corresponding to a rule base r are instantiated according to the constants appearing in r. This is the mechanism that implements scoped negation-as-failure (for a possible implementation see for instance [8]). The syntax necessary to explicitly declare predicate domains will be described in a subsequent paper, but basically it gets translated to domain predicates in the bodies of rules in order to guarantee correct instantiation of variables in rules (e.g. by using `rdf:type`, `rdf:domain` and `rdf:range` properties).

5 Comparison and Conclusions

The notion of localized closed world assumptions has been proposed for instance in [11]. The idea is to have syntactic mechanisms in the Semantic Web languages (like DAML+OIL or OWL) to express that a predicate is closed, i.e. something which cannot be inferred can be assumed false: this is a usual assumption in logic

programming (negation-as-failure, by default, or weak) and relational databases (the set difference operation of relational algebra). The major problem with the proposal of Heflin and Munoz-Avila is the use of a Clark's completion like approach, which is well-known to suffer from serious problems when applied to knowledge based systems [21, 20], even without negation.

The notion of scoped negation-as-failure has also been suggested by several authors, see for instance [14, 16], and systems like FLORA-2 [24] do support it. Both FLORA-2 and TRIPLE [22] support modularity constructions, which are essential for deployment of inference engines in the Semantic Web. Alternative proposals are already present in the `dlvhex` system [8], where the reader can find detailed discussion about applications to Semantic Web. This answer-set programming system has features like high-order atoms and external atoms which are very flexible. For instance, closure rules similar to our ones are expressed with high-order statements of the form

$$C'(X) \leftarrow o(X), \mathit{concept}(C), \mathit{concept}(C'), \mathit{cwa}(C, C'), \sim C(X)$$

where $\mathit{concept}(C)$ is a predicate which holds for all concepts C, $\mathit{cwa}(C, C')$ states that C' is the complement of C under the closed world assumption, and $o(X)$ is a predicate that holds for all individuals occurring in the knowledge base.

However, in contradistinction to the existing systems, we define the notion of objective, open and closed predicates, their semantically compatible definition, as well as languages constructs for controlling knowledge in the Semantic Web. The combination of open-world and closed-world reasoning in the same framework is also proposed in [2], where the ERDF stable model semantics of Extended RDF knowledge bases is developed, based on partial logic [12]. However, modularity issues are not considered there. The existence and combination of all our proposed mechanisms in a single language is a novelty, to the best of our knowledge.

The language is intuitive to use and gives absolute freedom to producers and consumers of knowledge in the Semantic Web. It can be implemented with the existing technology, and can support and integrate different inference engines ranging from relational databases to state-of-the-art inference engines, including description logic reasoners. Both tractable and more complex forms of inference are also easily syntactically identified and delimited. The semantics of the constructs can be defined via immediate program transformations, for which the rationale and corner-stone elements have been introduced in this paper.

There are still some important practical problems to be addressed at the implementation level for which solutions exist, but for lack of space cannot be presented in this work. Furthermore, the issue of contradiction is not addressed here, but the results of Section 2 can be adapted for existing paraconsistent semantics for extended logic programs, namely [1, 6, 19]. A prototypical implementation is underway, using immediate extensions to RuleML markup language [17].

References

1. J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14(1):93–147, 1995.

2. A. Analyti, G. Antoniou, C. V. Damásio, and G. Wagner. Stable Model Theory for Extended RDF Ontologies. In *4th Int. Semantic Web Conf.*, pages 21–36, 2005.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.
4. M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 12(1), 1993.
5. CWM-closed world machine. (`http://www.w3.org/2000/10/swap/doc/cwm.html`).
6. C. V. Damásio and L. M. Pereira. A survey of paraconsistent semantics for logic programs. In D. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, pages 241–320. Kluwer, 1998.
7. Duerst and Suignard. Internationalized Resource Identifiers. RFC 3987, Jan. 2005.
8. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proc. of IJCAI'05*, pages 90–96. Professional Book Center, 2005.
9. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *Proc. of 1st ESWS-2004*, pages 342–356, 2004.
10. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *Proc. of 7th ICLP*, pages 579–597. MIT Press, 1990.
11. J. Heflin and H. M. Avila. Lcw-based agent planning for the semantic web. In *Ontologies and the Semantic Web*, WS-02-11, pages 63–70. AAAI Press, 2002.
12. H. Herre, J. Jaspars, and G. Wagner. Partial Logics with Two Kinds of Negation as a Foundation of Knowledge-Based Reasoning. In D. M. Gabbay and H. Wansing, editors, *What Is Negation?* Kluwer Academic Publishers, 1999.
13. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004.
14. M. Kifer, J. de Bruijn, H. Boley, and D. Fensel. A realistic architecture for the semantic web. In *Proc. of RuleML 2005*, pages 17–29. Springer, November 2005.
15. L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *ECAI'92*, pages 102–106. John Wiley & Sons, 1992.
16. The rule interchange WG charter. (`http://www.w3.org/2005/rules/wg/charter`).
17. The Rule Markup Initiative (RuleML). Available at `http://www.ruleml.org`.
18. K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *Proc. of SIGMOD 1994 Conference*. ACM, 1994.
19. C. Sakama and K. Inoue. Paraconsistent Stable Semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.
20. J. Shepherdson. Negation in logic programming for general logic programs. In J. Minker, editor, *Found. of Ded. Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, 1988.
21. J. C. Shepherdson. Negation as failure: a comparison of Clark's completed data base and Reiter's CWA. *Journal of Logic Programming*, 1(1):51–79, 1984.
22. M. Sintek and S. Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *First International Semantic Web Conference on The Semantic Web (ISWC2002)*, pages 364–378. Springer-Verlag, 2002.
23. G. Wagner. Web Rules Need Two Kinds of Negation. In *Proc. of PPSWR'03*. Springer-Verlag, December 2003.
24. G. Yang, M. Kifer, and C. Zhao. Flora-2: A Rule-Based Know. Representation and Inference Infrastructure for the Sem. Web. In *ODBASE'03*, pages 671–688, 2003.