

Open and Closed Reasoning in the Semantic Web

Carlos Viegas Damásio (cd@di.fct.unl.pt)
Centro de Inteligência Artificial,
Universidade Nova de Lisboa,
Caparica, Portugal

DRAFT August 2005

1 Some problems and proposals for the Semantic Web

The Semantic Web brings new problems for the knowledge representation community, due to its distributed and world-wide character. At the time of the start of the Semantic Web Initiative it was clearly marked that the globalization of Knowledge Representation introduces new challenges:

The Semantic Web is what we will get if we perform the same globalization process to Knowledge Representation that the Web initially did to Hypertext. We remove the centralized concepts of absolute truth, total knowledge, and total provability, and see what we can do with limited knowledge. [8]

In the same set of documents [6], the following fundamental theoretical problems have been identified (besides other ones):

- Negation, Contradiction and Inconsistencies
- Open World versus Closed World assumptions
- Rule Systems in the Semantic Web

For the time being, the first two issues have been circumvented by discarding the facilities to introduce them, namely classical negation and closed world assumptions in RDF(S) [20]. The widely recognized need of having rules in the Semantic Web [23, 32] restarted the discussion of the fundamentals of closed-world reasoning in the Semantic Web, and the appropriate mechanisms in Rule Systems to implement it, and in particular “negation as failure.”

The RDF(S) recommendation [20] has been a major breakthrough, and provides solid ground to discuss the issues. We defend that partial logics [2, 38, 36, 29, 30, 27, 28, 22] are a step through, and a necessary one, for a powerful and expressive Semantic Web. Furthermore, we claim that semantics like

the ones proposed in the logic programming and deductive databases communities [17, 18, 36, 24, 30, 1, 33, 11, 13, 14, 28, 5] can be immediately used for specifying Rule Systems in the Semantic Web, in particular the one we have adopted in this paper [22].

2 Negation, Contradictions and Inconsistencies

The ability to express negative knowledge, besides positive one, is an obvious requirement for a knowledge representation language. This is particularly sensible in the Semantic Web, since for instance one has to be capable of expressing that a user/machine is “NOT authorized” to access some information. One initial justification for having this limitation appears in the Semantic Web Roadmap [7]:

As far as mathematics goes, the language at this point has no negation or implication, and is therefore very limited. Given a set of facts, it is easy to say whether a proof exists or not for any given question, because neither the facts nor the questions can have enough power to make the problem intractable. [7]

The tractability argument can no longer be used, since as it is stated in the RDF Semantics W3C recommendation [20], the general problem of determining simple entailment between arbitrary RDF graphs is decidable but NP-complete. So, the question is now whether there are any knowledge representation frameworks with the same, or similar complexity classes, supporting rules and (two kinds of) negation.

By restricting the language of our theories to the datalog case (no function symbols), the data complexity of the language proposed in the paper is co-NP-complete (i.e. testing if a query holds in all stable models), and NP-complete for testing the existence of a stable model. This is immediate from the bunch of complexity results for logic programming [15] and the relationship [22] of our semantics with (Paraconsistent) Answer Set Semantics [18, 26, 33]. Other semantics even have lower data complexity classes, like Well-founded Semantics [16] and its extensions supporting two forms of negation [1, 11, 13] which are P-complete. A major reason for these good complexity results is that the arrow symbol (\leftarrow) in our rules is interpreted as a sequent, not implication, and therefore they can only be used in one direction. The classical *Modus Tollens* cannot be applied, avoiding the indirect inference of disjunctive conclusions. Furthermore, disjunctive heads are also not allowed, thus it is easy to see that there is no way of obtaining arbitrary disjunctive conclusions. This language limitation keeps data complexity from increasing to higher complexity classes [15].

The existence of negation also paves the way to the problem of introducing contradictions in the Semantic Web, and is probably the main motivation for not having negation in RDF. This is an important concern, since the existence of a single contradiction in a rulebase, asserted or deduced, explodes the consequences and therefore trivializes the rulebase because of the validity of the *ex*

contradictione sequitur quodlibet principle in classical logic

$$\forall \Gamma \forall A \forall B \Gamma, A, \neg A \models B$$

where Γ is an arbitrary first-order theory, and A and B arbitrary formulae. In general terms, this means that if a single contradiction is present then everything can be concluded, rendering the knowledge useless.

The trivialization problem is particularly pertinent, since a contradiction will surely exist in the knowledge asserted in the whole Semantic Web: considering the Semantic Web as the union of all existing knowledge expressed in it is useless and unpractical. It can be argued that this is a good reason for not having negation, but notice that the trivialization problem already occurs in the proposed semantics for rdfs-entailment [20]. Since there are already rdfs-inconsistent graphs present in the Semantic Web (see <http://www.w3.org/2000/10/rdf-tests/rdfcore/rdfs-entailment/t>) adopting the omniscient view of the Web is senseless, and independently of the existence of negation: you can already conclude everything, if you accept rdfs-entailment. So, our understanding is that each user/agent/application should be concerned solely with the knowledge that decided/trusted/agreed to assert or load in its local subset of the Semantic Web.

Moreover, the introduction of negation does not necessarily force the adoption of *ex contradictione sequitur quodlibet*. There has been a huge theoretical work in paraconsistent logics from which the Semantic Web might benefit (see for instance [9]), and already has been in several proposals for logic programming knowledge representation systems with two kinds of negation which tolerate inconsistency [37, 26, 39, 1, 33, 11, 13, 14, 5]. The partial logic based semantics we propose here is paraconsistent and tolerates contradictions in the knowledge base: in our opinion, a contradiction should not always render all the rulebase useless.

Consider the case that your rules entail that a user a should be authorized to enter the system, as well as that this access should be denied. Does that affect the authorization of user b (for instance the system administrator) to access the system ? Of course, it can be said that it is your system that is ill-defined, but contradictions will definitely occur in a distributed and global Semantic Web. What shall be done ? Your system should be immediately useless, or can it be used meanwhile problems are solved/understood ? Clearly, this depends on the particular situation but both approaches should be permitted.

Consider the following knowledge, abstracting recent events in the World:

```

alive(mrA) ←
¬alive(mrA) ←
bury(?X) ← ¬alive(?X)
elected(mrB) ←
oil_price_increases ← elected(mrB)
oil_price_increases ← ¬alive(mrA)
teacher(mrC) ←

```

We have a contradiction between the facts $\neg alive(mrA)$ and $alive(mrA)$. Obviously, one should infer that mrB was elected and that mrC is a teacher, as

well as that that *oil_price_increases* is true. However, the conclusion that *mrA* is to be buried depends on contradiction, and its consequences should be taken with some care since it might be in fact alive. If we adopt the classical logic based semantics, we will also obtain that $\neg teacher(mrC)$, $\neg elected(mrB)$, and $\neg oil_price_increases$ because of the contradiction. These are to us unappropriate, and debatable conclusions.

The language and semantics we propose here is paraconsistent and tolerates contradictions, since it is intimately related to Nelson's constructive logic system N^- , and has well studied mathematical properties (see for instance [2, 22]). We believe that our partial logics based approach can be used to provide more general logical basis for the Semantic Web. This is according to the intuitions and remarks appearing in the LBase Working Group note [19]:

In this document, we use a version of first order logic with equality as Lbase. This imposes a fairly strict monotonic discipline on the language, so that it cannot express local default preferences and several other commonly-used non-monotonic constructs. We expect that as the Semantic Web grows to encompass more and our understanding of the Semantic Web improves, we will need to replace this Lbase with more expressive logics. However, we expect that first order logic will be a proper subset of such systems and hence we will be able to smoothly transition to more expressive Lbase languages in the future.

By imposing all predicates to be exact the classical first-order semantics is obtained from our partial logic model theory, since partial models will be isomorphic to classical ones. Syntactically, we can express that a predicate P must be exact by introducing the following two axioms in the theory:

$$\begin{array}{ll} P \vee \neg P & \text{totalness} \\ \sim P \vee \sim \neg P & \text{non-contradiction (or coherence)} \end{array}$$

Or this (uses implication, hiding disjunction...)

$$\begin{array}{ll} \sim P \supset \neg P & \text{totalness} \\ \neg P \supset \sim P & \text{non-contradiction (or coherence)} \end{array}$$

Or by introducing the following integrity constraints:

$$\begin{array}{ll} \leftarrow \sim P, \sim \neg P & \text{totalness} \\ \leftarrow P, \neg P & \text{non-contradiction (or coherence)} \end{array}$$

The advantage of using integrity constraints like the above is that the technique is not restricted solely to the current approach, but also applies to the most important semantics for deductive databases and logic programming with two kinds of negations [18, 33, 1, 11, 13]. In conclusion, the user/agent has a simple and modular mechanism for expressing that a predicate must be exact, and the syntactic machinery for expressing this in the accepted major semantics

is immediate. By not including any of the above, the user admits reasoning tolerant to contradiction.

Surely the problem of contradiction is controversial and potentially hazardous one, however it should not be neglected and not discussed just because it does not fit well the orthodoxy of classical logic (and has been a concern of logicians since the inception of this philosophical and mathematical endeavour [9]). Our logics and languages should be equipped with mechanisms to block the propagation of contradiction or simply warn the user/agent that some conclusion depends on the use of contradictory information, and even reason about it. There are already semantics and programming techniques that empower the applications with capabilities for performing safe reasoning with contradiction [26, 39, 1, 33, 14, 5].

3 Open World and Closed World assumptions

In the last years, we have assisted an intense quarrel about the benefits and problems of allowing nonmonotonic constructs for knowledge representation in the Semantic Web. It is now pretty clear that both sides of the dispute agree in the need of mechanisms for expressing nonmonotonic constructs, and the term “nonmonotonic” is pervasive in some Semantic Web related documents, some of which are collected below:

From RDF semantics recommendation [20]:

RDF is an assertional logic, in which each triple expresses a simple proposition. This imposes a fairly strict monotonic discipline on the language, so that it cannot express closed-world assumptions, local default preferences, and several other commonly used non-monotonic constructs.

From Lbase working group note [19]:

In this document, we use a version of first order logic with equality as Lbase. This imposes a fairly strict monotonic discipline on the language, so that it cannot express local default preferences and several other commonly-used non-monotonic constructs. We expect that as the Semantic Web grows to encompass more and our understanding of the Semantic Web improves, we will need to replace this Lbase with more expressive logics.

From SWRL proposal [23]:

Users also may want to restrict the expressiveness of the OWL classes and descriptions appearing in rules. One useful restriction on expressivity is Description Logic Programs [Grosz et al 2003] which, e.g., prohibits existentially-quantified knowledge in consequents. Suitably-restricted SWRL rules can be straightforwardly extended to enable procedural attachments and/or nonmonotonic reasoning (negation-as-failure and/or prioritised conflict handling) of the kinds supported in CCI rule systems and in RuleML

[RuleML] which facilitates interoperability between those CCI rule systems. Such adherence may thus facilitate combining SWRL knowledge with knowledge from those other rules languages. Suitable restrictions can also improve the empirical tractability of reasoning with rules.

We argue that a language with two kinds of negation is essential for satisfying both sides of the dispute, by providing an answer to the major objection to the use of nonmonotonic constructs as well as not losing expressive power:

The relationship between monotonic and nonmonotonic inferences is often subtle. For example, if a closed-world assumption is made explicit, e.g. by asserting explicitly that the corpus is complete and providing explicit provenance information in the conclusion, then closed-world reasoning is monotonic; it is the implicitness that makes the reasoning nonmonotonic. Nonmonotonic conclusions can be said to be valid only in some kind of 'context', and are liable to be incorrect or misleading when used outside that context. Making the context explicit in the reasoning and visible in the conclusion is a way to map them into a monotonic framework.[20]

We agree that the context should be made explicit, and go even further in that the use of open/closed world assumptions should be controlled by the publishers and consumers of knowledge in the Semantic Web: everyone should be able to express closed-world assumptions and no one should be obliged to accept them. We provide the mechanisms for this, depending on simple good practices of programming and of existence of two forms of negation. Clearly, this should be done in a way that respects the normative recommendations of RDF semantics, in particular by using a semantics which extends first-order logic. Noticeably, the approach we present in this work provides an encompassing solution as we have argued in the previous subsection.

The incompleteness of knowledge in the Semantic Web **IS** the motivation for using nonmonotonic constructs to infer new knowledge, and is mostly motivated by impossibility, both practical and theoretical, of constructing Semantic Web omniscient agents: every actor in the Semantic Web can only have access to a tiny part of the knowledge in the Semantic Web. If you adopt a strictly monotonic discipline you will get too sceptical in many situations. So, what mechanisms should be provided in order to extract more knowledge from your asserted knowledge and rulebases ?

First, ad hoc mechanisms like the ones proposed in the CWM system [10] (log:DefinitiveDocument and log:notIncludes) are not the correct way to go: these are hacker techniques which are not declarative and have without clean semantics (if at all). The nonmonotonic constructs studied and proposed by the Non Monotonic Reasoning, Deductive Databases and Logic Programming communities in the last 30 years are the **ONLY** sound way to proceed. The issues are subtle and took these communities a lot of time to reach consensus about the most valuable proposals and solutions. The solutions like the ones

in CWM will simply not work in full scale: what is the semantics of knowledge bases which include these statements and refer to each other ?

Local Closed World assumptions, like the ones proposed in [21] are much more interesting and an interesting way to pursue. 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 an 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 Hefflin 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 [35, 34], even without negation. For instance, if someone expresses the following knowledge:

$$\begin{aligned} memberEU(Austria) &\leftarrow \\ memberEU(Belgium) &\leftarrow \\ &\vdots \\ memberEU(UnitedKingdom) &\leftarrow \end{aligned}$$

By completing the previous knowledge, you will get that every country which is not listed is not a member state of the European Union, which is the intended meaning. However, if the following rule is added to the previous set of facts:

$$memberEU(?country) \leftarrow memberEU(?country)$$

then, the completion introduces a tautology which prevents the derivation of the intended negative conclusions; this is an undesired feature.

It is also known from the literature that in general the Closed World Assumption is not the same as Completion. In particular, it can easily become inconsistent when negation is introduced in the language [34]. This justifies our adoption of logic programming based semantics [17, 16] for providing interesting and generally adopted semantics for rules with negation(s), and are related to the major nonmonotonic reasoning forms (see for instance [4, 25]).

Our proposal consists in adding to the previous knowledge the following rule

$$\neg memberEU(?country) \leftarrow \sim memberEU(?country)$$

whenever it is known that *memberEU* predicate is closed. This will have the intended meaning for both previous situations, and in fact is accepted as the correct way of completing knowledge in logic programming languages with two kinds of negation, either based on well-founded semantics or in stable model semantics. Moreover, this is a very simple modular and **localized** mechanism to declare that a predicate is closed, with understood behaviour and mathematical properties as well as widely accepted proper semantics. Notice that this mechanism relies intrinsically on the existence of two forms of negation, like the ones adopted in this work. Symmetrically, one can complete negative knowledge like in the following example, which models a simple remote connection

authorization manager:

$$\begin{aligned}
&\neg authorize(root) \leftarrow \\
&\neg authorize(?user) \leftarrow \neg registered(?user) \\
&authorize(?user) \leftarrow \sim \neg authorize(?user) \\
®istered(a) \leftarrow \\
®istered(c) \leftarrow \\
®istered(root) \leftarrow \\
&\neg registered(?user) \leftarrow \sim registered(?user)
\end{aligned}$$

Remote connections to user *root* are not authorized, as well as to any user not registered in the system (the use of weak negation is fundamental in order to avoid listing **ALL** the unregistered users). Users *a*, *c* and *root* are registered users. In particular, we are able to conclude from the above knowledge that:

$$\begin{array}{ll}
\neg authorize(root) & authorize(c) \\
\neg authorize(b) & authorize(a)
\end{array}$$

Notice that $\neg authorize(?user)$ holds for any *?user* distinct from *a* and *c*, in particular for *b*.

The practical significance of the proposed mechanism is that the user/knowledge engineer may construct Semantic Web applications' knowledge bases using solely the strong negation connective (\neg), and then explicitly declaring the predicates which are closed, on an individual per individual basis. There is no loss of generality with this approach, since a weak negation can always be introduced by resorting to an auxiliary predicate and closing a program rule.

Suppose, that a Semantic Web programmer wants to use weak negation in his knowledge bases. He introduces a new predicate symbol, say *not_P* to represent the weak negation of *P*, which can be defined by closing the negative instances of the rule $\neg not_P \leftarrow P$. A similar technique can be applied to obtain the weak negation of $\neg P$, captured by predicate *not_neg_P*:

$$\begin{array}{ll}
\neg not_P \leftarrow P & \neg not_neg_P \leftarrow \neg P \\
not_P \leftarrow \sim \neg not_P & not_neg_P \leftarrow \sim \neg not_neg_P
\end{array}$$

Furthermore, one can even relate both forms of negation by introducing an extra rule, in the spirit of the semantics proposed in [31, 1, 11, 13] obeying to the coherence principle, by letting strong negation entail the weak form:

$$\begin{array}{ll}
\neg cnot_P \leftarrow P & \neg cnot_neg_P \leftarrow \neg P \\
cnot_P \leftarrow \neg P & \neg cnot_neg_P \leftarrow P \\
cnot_P \leftarrow \sim \neg not_P & cnot_neg_P \leftarrow \sim \neg cnot_neg_P
\end{array}$$

The major distinction between both forms is that, in face of contradiction between *P* and $\neg P$ one also obtains *cnot_P* and *cnot_neg_P*, i.e. a localized explosion occurs. This property is used in Paraconsistent Well-founded Semantics with Explicit Negation to detect dependencies on contradiction [1, 11, 13, 5].

We conclude from the above discussion that strong negation plus localized closures have the same expressive power as strong negation plus weak negation:

$$RULES(\neg, LCLOSURE) \equiv RULES(\neg, \sim)$$

The discussion of the advantages of one mechanism over the other, are in some sense futile. If you have one, you can get the other. There are still some other issues to address related to the use of weak negation, namely non-ground weak negations, which have been studied in the literature and are fully understood, but lie outside the scope of this paper.

4 Expressing Context in RuleML

In this section we study the syntactical mechanisms in order to be able to express the necessary context to use strong and weak negations safely in the Semantic Web environment. We focus the discussion in the RuleML proposal [32] since it already allows several forms of negation, but lacks controlled ways of using them in the Semantic Web environment. When defining knowledge bases for begin used in the Semantic Web, we have to take into account the following four levels of context and their interaction:

- The Semantic Web context;
- The application context, corresponding to the context where a user or Semantic Web agent loads or asserts the relevant, or consumes the knowledge provided by inference engines in the Semantic Web;
- The knowledge 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;

Knowledge 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 be always equivalent to loading it locally, 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 they are declaring cannot be defined elsewhere, and may declare local 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 knowledge bases. It is mandatory that, by default, all knowledge inferred from the combination of several knowledge bases and asserted facts into the application context must be safe in the Semantic Web.

The challenge is to provide simple mechanisms in order to guarantee the fulfilment of the previous requirements. The syntax of RuleML should be extended in order to declare how relations (predicates) are made visible, or hidden. It is also necessary to express how external information to the knowledge base is incorporated into it. We propose the following two roles of the `Rulebase` element for achieving this:

declares: This role specifies which predicates are declared in the knowledge base, and its content contains `Rel` elements with the optional new attributes `arity`, `visibility`, `values`, `context`, and `dynamic`.

uses: This role specifies which predicates are used from other knowledge bases or from the Semantic Web, and its content contains `Rel` elements with the optional new attribute `arity`, and `from`.

In order to be usable in the Semantic Web, the `rbaselab` of the top-level element `Rulebase` must be declared with an `uri` attribute. The knowledge base developer specifies the visibility of the predicates defined in the knowledge base with the `declares` role. The `Rel` element identifies the relation that it is being declared. The `arity` attribute contains a list of natural numbers identifying the arities (number of arguments) of the predicates being declared; if absent, the predicate is being declared with arbitrary arity. The `values` attribute specifies the allowed truth-values for the predicate. Currently, the truth-values allowed values are "boolean" or "exact", "belnap" or "four", "total" and "coherent", the default being "four". The `visibility` attribute 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:

"open": a predicate declared open is visible outside the knowledge base, and intends to capture global predicates which are being defined in the Semantic Web. In particular, it can be declared elsewhere and only strong negation \neg can be used. These predicates must have also an `uri` attribute, specifying its global name. Additionally, the attribute `context` can specify a list of URIs where the predicate can be used; if absent, it can be used everywhere.

"visible": these predicates are visible outside the knowledge bases where have been declared, but cannot be declared by any other knowledge base in the Semantic Web. As before, only strong negation can be applied to these predicates, and `uri` should be used to specify its global name. No predicate in the RDF and RDFS vocabularies can be declared visible. As in the previous situation, the attribute `context` can specify a list of URIs where the predicate can be used; if absent, it can be used everywhere.

"local": these are predicates are to be used only inside the knowledge base, but are restricted in that only \neg can be used.

"**closed(pos)**" or "**closed(neg)**": these are predicates for use only in the current module but rules are completed. If "**closed(pos)**" is used to declare predicate P then the rule $\neg P \leftarrow \sim P$ is implicitly asserted, and the dual one for "**closed(neg)**", completing the knowledge. Again, anywhere else in the knowledge base, only strong negation can be used. The closure rule may be inhibited by the consumer of the knowledge.

"**internal**": predicate is internal to the knowledge base but now any form of negation can be used inside the knowledge base, and its use can be forbidden by the consumer of the knowledge.

The **Rel** element inside declares roles might be complemented with an attribute **dynamic** (taking values "**true**" or "**false**") stating whether a predicate is dynamic or static. Rules or facts of dynamic predicates might be added or removed by the agent using the knowledge. By default, a predicate is assumed static.

The **uses** role is easier to employ, and simply states which predicates are defined out of the scope of the current knowledge base and which should be "imported" into the knowledge base. The **arity** attribute works as before. The **from** attribute specifies the source knowledge bases where the predicate can be found, which is a list of URIs (or in the future, IRIs). A knowledge base can only use predicates if allowed by the **context** attribute in the corresponding **declares** elements. If this attribute is omitted, then the predicate is being imported from every available knowledge bases in the current application context. All the **Rel** elements occurring inside a **uses** role must have an **uri** attribute.

All predicates with an **uri** attribute, which are used inside the knowledge base and not mentioned neither in the **declares** nor **uses** assertions, are by default declared open and usable from anywhere. Otherwise, they are local.

The interaction of these mechanisms are controlled by the user or application controlling a local context. Therefore, it is necessary to specify some requirements of the API which loads, asserts or links knowledge in the local context. No syntax is currently provided since it is outside of the scope of the RuleML language, i.e. it is at the interface/command-line level. The agent should be able to use any of the following modes:

allow-all: In this mode, any form of reasoning is allowed, monotonic or not. The only error that might occur is that a predicate declared visible in some module is declared elsewhere. In this case, the agent may pick one of the knowledge bases to use or abort. Inconsistencies and contradictions are detected, and the user appropriately warned. Consequences obtained by default reasoning might also be identified. This also applies to the subsequent modes.

allow-closed: In this mode, locally closed predicates might be used, but no predicate can be declared internal. If there is an internal predicate, the knowledge base cannot be used, and processing should abort.

force-open: In this mode, closure rules are inhibited and cannot be used in the inference engine by its deductive mechanisms. Obviously, internal

predicates are forbidden.

reject-closed: The final mode rejects both closed and internal predicates in knowledge bases.

We suggest to use **force-open** as default mode, since it does not extract consequences by nonmonotonic reasoning, and knowledge bases might be combined together monotonically (if the mode used is always **force-open**). Furthermore, all conclusions obtained with this mode are also obtained in mode **allow-closed**. However, some consequences obtained in any of the modes other than **force-open** might not be inferred in **allow-closed**. So, any producer/consumer may safely use knowledge bases in the Semantic Web under **force-open** mode. Mode **allow-closed** can be used to extract more consequences, but which can be retracted later on when new knowledge is made available.

As a final remark, the agent requires mechanisms to assert knowledge into an existing knowledge base or declare that in its context, a open predicate can be considered closed.

5 Acknowledgments

This work has benefitted from discussions with Anastasia Analyti, Grigoris Antoniou and Gerd Wagner from the REVERSE (<http://www.reverse.net>) Network of Excellence. I also thank Lus Moniz Pereira and Jos Jlio Alferes for their detailed comments. Parts of this paper appear in a revised form in [3].

Referências

- [1] J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Special Issue of the Journal of Automated Reasoning*, 14(1):93–147, 1995.
- [2] A. Almukdad and D. Nelson. Constructible falsity and inexact predicates. *JSL*, 49:231–233, 1984.
- [3] A. Analyti, G. Antoniou, C. V. Damásio, and G. Wagner. Negation and negative information in the w3c resource description framework. *Annals of Mathematics, Computing & Teleinformatics*, 2(1):25–34, 2004.
- [4] G. Antoniou. *Nonmonotonic Reasoning*. MIT Press, 1997.
- [5] J. ao Alcântara, C. V. Damásio, and L. M. Pereira. Paraconsistent logic programs. In S. Flesca and G. Ianni, editors, *Logics In Artificial Intelligence, 8th European Conference, JELIA2002*, LNAI 2424, pages 345–356. Springer, 2002.
- [6] T. Berners-Lee. Design issues - architectural and philosophical points. Personal notes, 1998. Available at <http://www.w3.org/DesignIssues/>.

- [7] T. Berners-Lee. Semantic web road map. Personal notes, 1998. Available at <http://www.w3.org/DesignIssues/Semantic.html>.
- [8] T. Berners-Lee. What the semantic web can represent. Personal notes, 1998. Available at <http://www.w3.org/DesignIssues/RDFnot.html>.
- [9] W. A. Carnielli and J. ao Marcos. A taxonomy of c-systems. *Lecture Notes in Pure and Applied Mathematics*, 228, 2002.
- [10] Cwm - closed world machine. Available at <http://www.w3.org/2000/10/swap/doc/cwm.html>.
- [11] C. V. Damásio. *Paraconsistent Logic Programming with Constraints*. PhD thesis, Faculdade de Ciências e Tecnologia, Lisboa, Outubro 1996. 375 páginas.
- [12] C. V. Damásio. Open and closed reasoning in the semantic web. Technical report, Centro de Inteligência Artificial da Universidade Nova de Lisboa, 2005. Available from <http://centria.di.fct.unl.pt/~cd>.
- [13] C. V. Damásio and L. M. Pereira. A paraconsistent semantics with contradiction support detection. In J. Dix, U. Furbach, and A. Nerode, editors, *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 224–243, Castelo de Dagstuhl, Alemanha, July 1997. Springer.
- [14] C. V. Damásio and L. M. Pereira. A survey of paraconsistent semantics for logic programas. In D. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, Reasoning with Actual and Potential Contradictions. Coordenado por P. Besnard e A. Hunter, pages 241–320. Kluwer Academic Publishers, 1998.
- [15] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
- [16] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
- [17] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
- [18] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.
- [19] R. V. Guha and P. Hayes. Lbase: Semantics for languages of the semantic web. W3C Note, 10 October 2003. Available at <http://www.w3.org/TR/2003/NOTE-lbase-20031010/>.

- [20] P. Hayes. Rdf semantics. W3C Recommendation, 10 February 2004. Available at <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>.
- [21] J. Heflin and H. Munoz-Avila. Lcw-based agent planning for the semantic web. In *Ontologies and the Semantic Web*, Papers from the 2002 AAAI Workshop WS-02-11, pages 63–70. AAAI Press, 2002.
- [22] H. Herre, J. Jaspars, and G. Wagner. *What is Negation ?*, chapter Partial Logics with Two Kinds of Negation as a Foundation for Knowledge-Based Reasoning. Kluwer Academic Publishers, 1999.
- [23] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available at <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.
- [24] R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*. MIT Press, 1990.
- [25] V. W. Marek and M. Truszczyński. *Nonmonotonic Logic*. Springer-Verlag, 1993.
- [26] D. Pearce. Reasoning with Negative Information, II: hard negation, strong negation and logic programs. In D. Pearce and H. Wansing, editors, *Non-classical logic and information processing*, number 619 in LNAI, pages 63–79. Springer-Verlag, 1992.
- [27] D. Pearce. Answer sets and constructive logic, II: Extended logic programs and related nonmonotonic formalisms. In L. Pereira and A. Nerode, editors, *Logic Programming and Nonmonotonic Reasoning - proceedings of the second international workshop*, pages 457–475. MIT Press, 1993.
- [28] D. Pearce. Stable inference as intuitionistic validity. *Journal of Logic Programming*, 38(1):79–91, 1999.
- [29] D. Pearce and G. Wagner. Reasoning with negative information I: Strong negation in logic programs. In L. Haaparanta, M. Kusch, and I. Niiniluoto, editors, *Language, Knowledge and Intentionality*, pages 430–453. Acta Philosophica Fennica 49, 1990.
- [30] D. Pearce and G. Wagner. Logic programming with strong negation. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*, pages 311–326. LNAI 475, Springer-Verlag, 1991.
- [31] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, 1992.
- [32] The rule markup initiative (ruleml). Available at <http://www.ruleml.org>.

- [33] C. Sakama and K. Inoue. Paraconsistent Stable Semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.
- [34] J. Shepherdson. Negation in logic programming for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 19–88. Morgan Kaufmann, 1988.
- [35] J. C. Shepherdson. Negation as failure: a comparison of Clark’s completed data base and Reiter’s Closed World Assumption. *Journal of Logic Programming*, 1(1):51–79, 1984.
- [36] G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H.-D. Gerhardt, editors, *Mathematical Foundations of Database Systems*, pages 357–371. LNCS 495, Springer–Verlag, 1991.
- [37] G. Wagner. Ex contradictione nihil sequitur. In *International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.
- [38] G. Wagner. Logic programming with strong negation and inexact predicates. *Journal of Logic and Computation*, 1(6):835–859, 1991.
- [39] G. Wagner. Reasoning with inconsistency in extended deductive databases. In L. M. Pereira and A. Nerode, editors, *2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 300–315. MIT Press, 1993.