Negation and Negative Information in the W3C Resource Description Framework*

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

[1] Institute of Computer Science, ICS-FORTH, Greece
[2] Department of Computer Science, University of Crete, Greece
[3] Centro de Inteligência Artificial, Universidade Nova de Lisboa, Caparica, Portugal
[4] Department of Information Systems, Eindhoven University of Technology, The Netherlands
analyti@ics.forth.gr, antoniou@ics.forth.gr,
cd@di.fct.unl.pt, G.R.Wagner@tm.tue.nl

Abstract. The concept of negation plays a special role in non-classical logics and also in knowledge representation formalisms where negative information has to be taken into account on par with positive information. In the tradition of mathematical logic, there is a general preference to consider positive information as basic and treat negative information as derived. This has also been the approach in relational databases, in normal logic programs, and is now again the approach in the Resource Description Framework (RDF) that has recently been proposed as a general language for representing propositional information on the Web by the World Wide Web Committee (W3C). However, as we argue in this article, any practical knowledge representation formalism, especially for the Web, has to be able to deal with knowledge items involving partial predicates for which negative information is as informative as positive information, and which may have truth-value gaps and truth-value clashes. This kind of knowledge is best represented and processed with the help of the two negations of partial logic, one expressing explicit falsity and the other one expressing non-truth.

1 Introduction

Due to its distributed and world-wide nature, the Web creates new problems for knowledge representation research. Already when the Semantic Web Initiative was launched it was observed that the globalization of knowledge representation introduces new challenges:

* 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 (www.rewerse.net). Some ideas described in this paper have been presented by Gerd Wagner at the International Workshop on Negation in Constructive Logic, July 1–4, 2004, Dresden University of Technology, Germany.

> *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. [6]*

In the same set of documents [4], the following fundamental theoretical problems have been identified (besides other ones):

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

For the time being, the first two issues have been circumvented by discarding the facilities to introduce them, namely negation and closed world assumptions in RDF(S) [16]. The widely recognized need of having rules in the Semantic Web [19, 26] 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 RDF(S) recommendation [16] has been a major step forward and provides solid ground to discuss the issues. We defend that partial logics [18] are fundamental for knowledge representation in general, and for knowledge representation on the Web in particular. Furthermore, we argue that semantics and inference operations like the ones proposed in the logic programming and deductive database communities [13, 14, 30, 20, 24, 1, 27, 8–10, 23, 3] should be the basis for developing Web rule formalisms.

1.1 Partial Logic Semantics for Computational Forms of Negation

In [30], it was argued that a database, as a knowledge representation system, needs two kinds of negation to be able to deal with partial information. In [33], this point was made for the Semantic Web as a framework for knowledge representation in general, and in the present paper we make the same point for the Semantic Web language RDF and show how it can be extended to accommodate the two negations of partial logic.

Computational forms of negation are used in imperative programming languages (such as *Java*), in database query languages (such as *SQL*), in modeling languages (such as *UML/OCL*), in production rule systems (such as *CLIPS* and *Jess*) and in logic programming languages (such as *Prolog*). In imperative programming languages, negation may occur in the condition expression of a conditional branching statement. In database query languages, negation may occur in at least two forms: as a *not* operator in selection conditions, and in the form of the relational algebra *difference* operator (corresponding to the SQL *EXCEPT* operator). In modeling languages, negation occurs in constraint statements. E.g., in OCL, there are several forms of negation: in addition to the *not* operator in selection conditions also the *reject* and the *isEmpty* operators are

used to express a negation. In production rule systems, and in logic programming languages, a negation operator *not* typically occurs only in the condition part of a rule with the operational semantics of *negation-as-failure* which can be understood as classical negation under the preferential semantics of stable models.

In all these computational information processing systems, negation is, from a logical point of view, not a clean concept, but combines classical (Boolean) negation with negation-as-failure and the strong negation of three-valued logic (also called *Kleene negation*). In any case, however, it seems to be essential for all these systems to provide different forms of negation.

In natural language, there are (at least) two kinds of negation: a weak negation expressing non-truth (in the sense of "she doesn't like snow" or "he doesn't trust you"), and a strong negation expressing explicit falsity (in the sense of "she dislikes snow" or "he distrusts you"). Notice that the classical logic *law of the excluded middle* holds only for the weak negation (either "she likes snow" or "she doesn't like snow"), but not for the strong negation: it does not hold that "he trusts you" or "he distrusts you"; he may be neutral and neither trust nor distrust you.

A number of knowledge representation formalisms and systems (see, e.g., [14, 30, 20, 1, 27, 10]) follow this distinction between weak and strong negation in natural language. However, many of them do not come with a model-theoretic semantics in the style of classical logic. Instead, an inference operation, that may be viewed as a kind of proof-theoretic semantics, is proposed.

Classical (two-valued) logic cannot account for two kinds of negation because two-valued (Boolean) truth functions do not allow to define more than one negation. The simplest generalization of classical logic that is able to account for two kinds of negation is *partial logic* giving up the classical bivalence principle and subsuming a number of 3-valued and 4-valued logics. For instance, in 3-valued logic with truth values $\{f, u, t\}$ standing for *false*, *undetermined* (also called *unknown* or *undefined*) and *true*, weak negation (denoted by $\sim$) and strong negation (denoted by $\neg$) have the following truth tables:

p	$\sim p$
t	f
u	t
f	t

p	$\neg p$
t	f
u	u
f	t

Notice the difference between weak and strong negation in 3-valued logic: if a sentence evaluates to u in a model, then its weak negation evaluates to t, while its strong negation evaluates to u in this model. Partial logics allow for *truth-value gaps* created by partial predicates to which the law of the excluded middle does not apply.

However, even in classical logic, where all predicates are total, we may distinguish between predicates that are completely represented in a database (or knowledge base) and those that are not. The classification if a predicate is completely represented or not is up to the owner of the database: the owner must

know for which predicates she has complete information and for which she does not. Clearly, in the case of a completely represented predicate, negation-as-failure amounts to classical negation, and the underlying completeness assumption is also called *Closed-World Assumption*. In the case of an incompletely represented predicate, negation-as-failure only reflects non-provability, but does not allow to infer the classical negation. Unfortunately, neither CLIPS/Jess nor Prolog support this distinction between 'closed' and 'open' predicates.

Open (incompletely represented total) predicates must not be confused with partial predicates that have truth-value gaps. The law of the excluded middle, $p \vee \neg p$, applies to open predicates but not to partial predicates.

For being able to make all these distinctions and to understand their logical semantics, we have to choose partial logic as the underlying logical framework. Partial logic allows to formally distinguish between falsity and non-truth by means of strong and weak negation. In the case of a total predicate, such as being an odd number, both negations collapse:

$$\sim odd(x) \text{ iff } \neg odd(x),$$

or in other words, the non-truth of the atomic sentence $odd(x)$ amounts to its falsity. In the case of a partial predicate, such as *likes*, we only have the relationship that the strong negation implies the weak negation:

$$\sim likes(she,snow) \text{ if } \neg likes(she,snow),$$

but not conversely. Also, while the double negation form '$\neg \sim$' collapses (according to partial logic, see [18]), the double negation form '$\sim \neg$' does not collapse: not disliking snow does not amount to liking snow. Classical logic can be viewed as the degenerate case of partial logic when all predicates are total.

1.2 The W3C Resource Description Framework (RDF)

RDF is a special predicate logical language that is restricted to conjunctive sentences (i.e. existentially quantified conjunctions of atomic formulas) involving binary predicates, only. Due to its purpose, RDF has a number of special features that distinguishes it from traditional logical languages:

1. It uses a special jargon, where the things of the universe of discourse are called *resources*, types are called *classes*, and binary predicates are called *properties*. Like binary relations in set theory, properties have a *domain* and a *range*. Resources are classified with the help of a *type* property (for stating that a resource is of type C, where C is a class).
2. It distinguishes two sorts of individuals: proper individuals and *literals* (or, more precisely, *literal values*), which are the denotations of lexical strings.
3. Properties are resources, that is, predicates are also elements of the universe of discourse. Consequently, it is possible to state properties of properties, i.e. make statements about predicates.

4. All properties and resources, except literals, are named with the help of a globally unique reference schema, called Uniform Resource Identifier (URI), that has been developed for the Web.
5. RDF comes with a non-standard model-theoretic semantics developed by Pat Hayes on the basis of an idea of Christopher Menzel, which allows self-application without violating the axiom of foundation. An example of this is the provable sentence stating that rdfs:Class, the class of all classes, is an instance of itself.

The predefined vocabulary of RDF comes in two layers:

1. the basic RDF layer, which includes the terms *type* and $Property$,
2. the RDF Schema (RDFS) layer, which includes the terms: *Resource, Literal, Class, Datatype, domain, range, subClassOf* and $subPropertyOf$.

1.3 Open World and Closed World Reasoning

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

From RDF semantics recommendation [16]:
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 [15]:
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 [19]:
Users also may want to restrict the expressiveness of the OWL classes and descriptions appearing in rules. [...] Suitably-restricted SWRL rules can be straightforwardly extended to enable procedural attachments and/or non-monotonic reasoning (negation-as-failure and/or prioritized conflict handling) [...]

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 against using nonmonotonic constructs as well as not loosing 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.[16]

We agree that considering the context may help to make nonmonotonic reasoning more transparent. One possible step in this direction is to allow publishers and users of Web knowledge making their own open or closed world assumptions about particular predicates.

We think that for each Web user/agent/application one should make a distinction between the 'local' knowledge under its own control, other local knowledge items controlled by other agents, and the 'public' knowledge items shared within communities. The latter is not controlled by a single agent but by entire communities.

The natural incompleteness of the knowledge available to a reasoning agent in many domains, in particular in the context of the Web, is the main reason to employ nonmonotonic constructs for jumping to conclusions needed for decision making and acting. Adopting a strictly monotonic reasoning discipline will not allow to draw any conclusion in many situations, so no decision can be made and no action performed, which is often simply not feasible. So, what mechanisms should be employed for deriving implicit knowledge from the knowledge items that have been asserted?

First, we may observe that ad-hoc constructs like the CWM operators *log:DefinitiveDocument* and *log:notIncludes* (see [7]) are certainly good for experimentation but do not provide a general approach. It seems to be more promising to investigate the nonmonotonic constructs studied and proposed by the Nonmonotonic Reasoning, Deductive Databases, and Logic Programming communities in the last 30 years. The issues are subtle and took these communities a lot of time to reach consensus about the most valuable proposals and solutions.

1.4 Contradictions and Inconsistencies

The ability to express negative knowledge, besides positive one, is an obvious requirement for a knowledge representation language. This also holds for the new languages of 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 [5]:

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. [5]

The tractability argument can no longer be used, since as it is stated in the RDF(S) W3C recommendation [16], 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 [11] and the relationship [18] of our semantics with (Paraconsistent) Answer Set Semantics [14, 22, 27]. Other formalisms even have lower data complexity classes, like the Well-founded Semantics [12] and its extensions supporting two forms of negation [1, 8, 9] 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, instead of an implication, and therefore it 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 [11].

The availability of negation also paves the way to the problem of handling contradictions in the Semantic Web. Avoiding contradictions by not allowing to express them is probably the main motivation for not having negation in RDF. This is an important concern, since in classical logic the existence of a single contradiction in a 'theory', or knowledge base, leads to an explosion of the consequence set because of the *ex contradictione sequitur quodlibet* principle

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

where Γ is an arbitrary first-order theory, and A and B arbitrary formulae. Logics based on this principle are also called *explosive*. In general terms, this means that if a single contradiction is present then everything can be concluded (every sentence is trivially true), rendering the entire body of knowledge useless.

However, contradictions will surely exist among the knowledge items asserted on the Web. Some may argue that in order to avoid contradictions, we should not have negation in Web languages, such as RDF. But notice that the inconsistency problem already occurs in RDFS even without negation. Since there are RDFS-inconsistent graphs on the Web (see, e.g., `http://www.w3.org/2000/10/rdf-tests/rdfcore/rdfs-entailment/test001.nt`), we can conclude everything according to the classical logic RDF(S) semantics. This is clearly unsatisfactory. The classical logic explosion principle does only make sense for mathe-

matical theories which claim to be true in the sense of metaphysics and which can therefore not tolerate any inconsistency. The situation is different in knowledge representation, where we do not deal with metaphysically true theories but with the beliefs (information and knowledge items) of fallible humans and software agents.

2 Open World and Closed World Reasoning

As opposed to the predicates such as 'likes' or 'is the author of', there are also predicates for which there is no need to express negative information because the available positive information about them is complete and, consequently, the negative information is simply the complement of the positive information.

For instance, the W3C has complete information about all official W3C documents and their normative status (http://www.w3.org/TR/ is the official list of W3C publications); consequently, the predicate *is an official W3C document* should be declared as closed in the W3C knowledge base (making a 'local' completeness assumption).[5] This consideration calls for a suitable extension of RDF(S) in order to allow making such declarations for specific predicates.

For sentences formed with closed predicates it is natural to use negation-as-failure for establishing their falsity: anything not listed on that page cannot be a W3C recommendation.

The concept of Local Closed World assumptions, as proposed in [17], is based on the idea to have syntactic mechanisms for being able to express that a predicate is closed, i.e. if it cannot be inferred to apply, then we can infer that its negation applies. The standard negation mechanism in logic programming (widely called *negation-as-failure*) is based on a general Closed World assumption. 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 (see [29, 28]), even without negation. For instance, if someone expresses the following knowledge:

$$\begin{array}{l} memberEU(Austria) \leftarrow \\ memberEU(Belgium) \leftarrow \\ \qquad\qquad \vdots \\ memberEU(UnitedKingdom) \leftarrow \end{array}$$

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:

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

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

[5] This example is due to Sandro Hawke.

It is also known from the literature that in general the Closed World Assumption is not the same as Completion, which can easily become inconsistent when negation is introduced in the language [28]. This justifies our adoption of logic programming based semantics [13, 12] for providing interesting and generally adopted semantics for rules with negation(s), and are related to the major nonmonotonic reasoning forms (see for instance [2, 21]).

A preferable approach is to add 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 inference or on stable model semantics. Moreover, this is a very simple modular and **localized** mechanism to declare that a predicate is closed, with well-understood behavior and mathematical properties as well as a widely accepted 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:

$$\begin{array}{l}
\neg authorize(root) \leftarrow \\
\neg authorize(?user) \leftarrow \ \neg registered(?user) \\
authorize(?user) \leftarrow \sim \neg authorize(?user) \\
registered(a) \leftarrow \\
registered(c) \leftarrow \\
registered(root) \leftarrow \\
\neg registered(?user) \leftarrow \sim registered(?user) \\
user(b) \leftarrow
\end{array}$$

Remote connections to the 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, whereas user b is not. 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 knowledge bases using only the strong negation connective ($\neg$), and then explicitly declaring the predicates which are closed, on an 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 [25, 1, 8, 9] obeying to the coherence principle, by letting strong negation entail the weak form:

$$\begin{array}{ll} \neg cnot\_P \leftarrow P & \neg cnot\_neg\_P \leftarrow \neg P \\ cnot\_P \leftarrow \neg P & cnot\_neg\_P \leftarrow P \\ cnot\_P \leftarrow\sim \neg not\_P & cnot\_neg\_P \leftarrow\sim \neg cnot\_neg\_P \end{array}$$

The major distinction between both forms is that, in the 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, 8, 9, 3].

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

3 Contradictions and Inconsistencies

Allowing negation in a knowledge representation language does not imply to adopt an explosive logic such as the classical logic semantics of RDF(S) [16]. There has been a lot of work on inconsistency-tolerant logics from which the Semantic Web may benefit. In particular, in the area of logic programming and knowledge representation with two kinds of negation, there are several proposals how to tolerate inconsistency (see [31, 22, 32, 1, 27, 8–10, 3]). Also the partial logic based semantics we propose here allows to tolerate contradictions in the knowledge base.

Consider the case that your rules entail that a user a is authorized to enter a certain information system, as well as that access should be denied. Should such a conflict affect the authorization of another 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 system such as the Web. What should be done in the face of such a contradiction? Your system could be immediately shut down and not turned on

again before it has been repaired, or you can continue to use it while the problem is being analyzed and solved? Clearly, the answer depends on the particular situation, but both approaches should be possible.

Consider the following knowledge, abstracted from recent events in the world:

$$
\begin{aligned}
&alive(mrA) \leftarrow \\
&\neg alive(mrA) \leftarrow \\
&bury(?X) \leftarrow \neg alive(?X) \\
&elected(mrB) \leftarrow \\
&oil\_price\_increases \leftarrow elected(mrB) \\
&oil\_price\_increases \leftarrow \neg alive(mrA) \\
&teacher(mrC) \leftarrow
\end{aligned}
$$

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 *oil_price_increases* is true. However, the conclusion that mrA is to be buried depends on contradictory information, and its consequences should be taken with some care since, in fact, he might be alive. If we adopt the explosion principle from classical logic, we will also obtain that $\neg teacher(mrC)$, $\neg elected(mrB)$, and $\neg oil\_price\_increases$ because of the contradiction, which is clearly nonsensical.

A partial logics based approach is able to provide a more general basis for the Semantic Web than the classical logic based approach of [16]. This is in line with the intuitions and remarks appearing in the LBase Working Group note [15]:

> *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 assuming that all predicates are total and coherent, the classical logic semantics is obtained from partial logic as a special case. Syntactically, we can express that a predicate P must be total and coherent by introducing the following two axioms in the theory:

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

Or with the help of implication instead of disjunction:

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

Or by introducing the following integrity constraints

$$\leftarrow\sim P, \sim\neg P \text{ totalness}$$
$$\leftarrow P, \neg P \qquad \text{non-contradiction (or coherence)}$$

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 [14, 27, 1, 8, 9]. In conclusion, the user/agent has a simple and modular mechanism for expressing that a predicate must be total and/or coherent, 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 handling is controversial and potentially hazardous, however it should not be a taboo just because it does not fit into the orthodoxy of classical logic. Our logics and languages should be equipped with mechanisms to block the propagation of contradictions 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 applications with capabilities for performing safe reasoning with contradiction [22, 32, 1, 27, 10, 3].

4 Extending RDF by Adding Negation and Partial Predicates

In this section, we extend RDF(S) by adding strong and weak negation. Additionally, we relax its semantics by allowing for partial predicates. For simplicity, we disregard literals, datatypes, RDF containers, collections, and reification, as these can be included by a straightforward extension.

A Web *vocabulary* V is defined to be a set of URI references. We denote the set of all URI references by URI. We consider a set Var of variable symbols such that $Var \cap URI = \emptyset$. We use the acronym *ERDF* for *Extended RDF*.

Definition 1 (ERDF triple). Let V be a vocabulary. A *positive ERDF triple* over V (also called *ERDF sentence atom*) is an expression of the form $p(s, o)$, where $s, o \in V \cup Var$ are called *subject* and *object*, respectively, and $p \in V$ is called *predicate* or *property*.
A *negative ERDF triple* over V is the strong negation $\neg p(s, o)$ of a positive ERDF triple $p(s, o)$ over V. An *ERDF triple* over V (also called *ERDF sentence literal*) is a positive or negative ERDF triple over V. □

We can also use the RDF-triple-like notation

$$s \quad -p \quad o\,.$$

for writing a negative ERDF triple and, as an option, use the + sign as a predicate prefix for marking positive triples, like in the following example:

ex:Gerd −ex:likes ex:CabernetSauvignon .
ex:Anastasia +ex:likes ex:CabernetSauvignon .
ex:Gerd +ex:likes ex:Riesling .
ex:Carlos −ex:likes ex:Riesling .

Definition 2 (ERDF formula). Let V be a vocabulary. We denote by $L(V)$ the smallest set that contains the positive ERDF triples over V, and is closed with respect to the following conditions: if $F, G \in L(V)$ then $\{\sim F,\ \neg F,\ F \wedge G,\ F \vee G\} \subseteq L(V)$. The connectives $\neg$, $\sim$ are called *strong negation* and *weak negation*, respectively. Additionally, we denote by $L(V|\neg, \wedge, \vee)$ the smallest set that contains the atomic ERDF triples over V, and is closed with respect to the following conditions: if $F, G \in L(V|\neg, \wedge, \vee)$ then $\{\neg F,\ F \wedge G,\ F \vee G\} \subseteq L(V|\neg, \wedge, \vee)$. Then, an *ERDF formula* over V is an element of $L(V)$, and a *persistent ERDF formula* over V is an element of $L(V|\neg, \wedge, \vee)$. □

Definition 3 (ERDF graph). An *ERDF graph* G is a set of ERDF triples over some vocabulary V. We denote the variables appearing in G by $Var(G)$. □

Let $G = \{t_1, ..., t_n\}$ be an $ERDF$ graph, and let $Var(G) = \{x_1, ...x_k\}$. Intuitively, G represents an existentially quantified conjunction of $ERDF$ triples. Specifically, G represents the formula $\exists x_1, ...x_k\ t_1 \wedge ... \wedge t_n$.

4.1 ERDF Model Theory

Definition 4. Partial interpretation
A *partial interpretation* I of a vocabulary V consists of:

- A set of things Res.
- A vocabulary interpretation function $I_V : V \rightarrow Res$.
- A set of properties $Prop \subseteq Res$.
- A property-truth extension function $PT_I : Prop \rightarrow \mathcal{P}(Res \times Res)$, and a property-falsity extension function $PF_I : Prop \rightarrow \mathcal{P}(Res \times Res)$.

We define $I(x) = I_V(x),\ \forall x \in V$. □

Definition 5. Satisfaction of an ERDF formula w.r.t. a partial interpretation and a valuation
Let F be an $ERDF$ formula and $Var(F)$ be the variables appearing in F. Let I be a partial interpretation of a vocabulary V. Let v be a mapping $v : Var(F) \rightarrow Res$ (called *valuation*). If $x \in Var(F)$, we define $[I + v](x) = v(x)$. If $x \in V$, we define $[I + v](x) = I(x)$.

- If $F = p(s, o)$ then $I, v \models F$ iff $< [I + v](s), [I + v](o) > \in PT_I(I(p))$.
- If $F = \neg p(s, o)$ then $I, v \models F$ iff $< [I + v](s), [I + v](o) > \in PF_I(I(p))$.
- If $F = \sim G$ then $I, v \models F$ iff all URIs appearing in G belong to V, and $I, v \not\models G$.
- If $F = F_1 \wedge F_2$ then $I, v \models F$ iff $I, v \models F_1$ and $I, v \models F_2$.
- If $F = F_1 \vee F_2$ then $I, v \models F$ iff $I, v \models F_1$ or $I, v \models F_2$.

– All other cases of $ERDF$ formulas are treated by the following DeMorgan-style rewrite rules expressing the falsification of compound ERDF formulas: $\neg(F \wedge G) \rightarrow \neg F \vee \neg G$, $\neg(F \vee G) \rightarrow \neg F \wedge \neg G$, $\neg\neg F \rightarrow F$, $\neg \sim F \rightarrow F$. $\square$

Let $V_{RDF} = \{rdf{:}type,\ rdf{:}Property\}$, and $V_{RDFS} = \{rdfs{:}Resource,\ rdfs{:}Class,\ rdfs{:}domain,\ rdfs{:}range,\ rdfs{:}subClassOf,\ rdfs{:}subPropertyOf\}$.

The *vocabulary of* $ERDF$, V_{ERDF}, is a set of URI references in the $erdf$: namespace. Specifically, $V_{ERDF} = \{erdf{:}TotalClass,\ erdf{:}TotalProperty,\ erdf{:}CoherentClass, erdf{:}CoherentProperty\}$. Intuitively, instances of the classes $erdf{:}TotalClass$ and $erdf{:}CoherentClass$ are classes that satisfy totalness and coherence, respectively. Similarly, instances of the classes $erdf{:}TotalPropery$ and $erdf{:}CoherentProperty$ are properties that satisfy totalness and coherence, respectively.

Definition 6. ERDF interpretation
An *ERDF interpretation* of a vocabulary V is a partial interpretation of $V \cup V_{RDF} \cup V_{RDFS} \cup V_{ERDF}$ extended by the new ontological categories $Cls \subseteq Res$ for classes, $TCls \subseteq Cls$ for total classes, and $TProp \subseteq Prop$ for total properties, $CCls \subseteq Cls$ for coherent classes, and $CProp \subseteq Prop$ for coherent properties, as well as the class-truth extension function $CT_I : Cls \rightarrow \mathcal{P}(Res)$, and the class-falsity extension function $CF_I : Cls \rightarrow \mathcal{P}(Res)$, such that:

1. $x \in Prop$ iff $< x, I(rdf{:}Property) >\in PT_I(I(rdf{:}type))$.
2. $x \in CT_I(y)$ iff $< x, y >\in PT_I(I(rdf{:}type))$, and $x \in CF_I(y)$ iff $< x, y >\in PF_I(I(rdf{:}type))$.
3. $Cls = CT_I(I(rdfs{:}Class))$.
4. $Res = CT_I(I(rdfs{:}Resource))$.
5. $< x, y >\in PT_I(I(rdfs{:}domain))$ and $< u, v >\in PT_I(x)$ implies $u \in CT_I(y)$.
6. $< x, y >\in PT_I(I(rdfs{:}range))$ and $< u, v >\in PT_I(x)$ implies $v \in CT_I(y)$.
7. $x \in Cls$ implies $< x, I(rdfs{:}Resource) >\in PT_I(I(rdfs{:}subclassOf))$.
8. $< x, y >\in PT_I(I(rdfs{:}subClassOf))$ implies $x, y \in Cls$ and $CT_I(x) \subseteq CT_I(y)$.
9. $PT_I(I(rdfs{:}subClassOf))$ is a reflexive and transitive relation on Cls.
10. $< x, y >\in PT_I(I(rdfs{:}subPropertyOf))$ implies $x, y \in Prop$ and $PT_I(x) \subseteq PT_I(y)$.
11. $PT_I(I(rdfs{:}subPropertyOf))$ is a reflexive and transitive relation on $Prop$.
12. $x \in TProp$ iff $< x, I(erdf{:}TotalProperty) > \in\ PT_I(I(rdf{:}type))$, and $x \in TProp$ implies $PT_I(x) \cup PF_I(x) = Res \times Res$.
13. $x \in TCls$ iff $< x, I(erdf{:}TotalClass) > \in\ PT_I(I(rdf{:}type))$, and $x \in TCls$ implies $CT_I(x) \cup CF_I(x) = Res$.
14. $x \in CProp$ iff $< x, I(erdf{:}CoherentProperty) > \in\ PT_I(I(rdf{:}type))$, and $x \in CProp$ implies $PT_I(x) \cap PF_I(x) = \emptyset$.
15. $x \in CCls$ iff $< x, I(erdf{:}CoherentClass) > \in\ PT_I(I(rdf{:}type))$, and $x \in CCls$ implies $CT_I(x) \cap CF_I(x) = \emptyset$.
16. I satisfies the RDF and $RDFS$ axiomatic triples, as well as the triples: $rdfs{:}subClassOf(erdf{:}TotalClass, rdfs{:}Class)$, $rdfs{:}subClassOf(erdf{:}TotalProperty, rdf{:}Property)$,

$rdfs{:}subClassOf(erdf{:}CoherentClass, rdf{:}Class)$,
$rdfs{:}subClassOf(erdf{:}CoherentProperty, rdfs{:}Property)$,

$rdf{:}type(X, erdf{:}CoherentProperty)$, where $X \in \{rdfs{:}domain, rdfs{:}range, rdfs{:}subClassOf,\ rdfs{:}subPropertyOf\}$,

$rdf{:}type(X, erdf{:}CoherentClass)$, where $X \in \{rdfs{:}Resource, rdf{:}Property, rdfs{:}Class\} \cup V_{ERDF}$. □

5 Conclusion

The basic language of the Semantic Web needs to accommodate two kinds of negation for representing and processing negative information involving truth-value gaps and truth-value clashes. For this reason RDF has to be extended in the way we have shown in this paper, and its classical logic semantics has to be refined into the partial logic semantics we have presented here. Without this extension, RDF will be too limited to capture the kind of information and knowledge items that occur in a globally distributed and decentralized knowledge space such as the Web.

References

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. G. Antoniou. *Nonmonotonic Reasoning*. MIT Press, 1997.
3. 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.
4. T. Berners-Lee. Design issues - architectual and philosophical points. Personal notes, 1998. Available at `http://www.w3.org/DesignIssues/`.
5. T. Berners-Lee. Semantic web road map. Personal notes, 1998. Available at `http://www.w3.org/DesignIssues/Semantic.html`.
6. T. Berners-Lee. What the semantic web can represent. Personal notes, 1998. Available at `http://www.w3.org/DesignIssues/RDFnot.html`.
7. Cwm - closed world machine. Available at `http://www.w3.org/2000/10/swap/doc/cwm.html`.
8. C. V. Damásio. *Paraconsistent Logic Programming with Constraints*. PhD thesis, Faculdade de Ciências e Tecnologia, Lisboa, Outubro 1996. 375 páginas.
9. 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.

10. 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.
11. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
12. 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.
13. 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.
14. 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.
15. 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/`.
16. P. Hayes. Rdf semantics. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.
17. 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.
18. H. Herre, J. Jaspars, and G. Wagner. Partial logics with two kinds of negation as a foundation of knowledge-based reasoning. In D. Gabbay and H. Wansing, editors, *What Is Negation?* Oxford University Press, 1999.
19. 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. Available at `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`.
20. R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*. MIT Press, 1990.
21. V. W. Marek and M. Truszczynski. *Nonmonotonic Logic*. Sppringer-Verlag, 1993.
22. D. Pearce. Reasoning with Negative Information, II: hard negation, strong negation and logic programs. In D. Pearce and H. Wansing, editors, *Nonclassical logic and information processing*, number 619 in LNAI, pages 63–79. Springer–Verlag, 1992.
23. D. Pearce. Stable inference as intuitionistic validity. *Journal of Logic Programming*, 38(1):79–91, 1999.
24. 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.
25. 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.
26. The rule markup initiative (ruleml). Available at `http://www.ruleml.org`.
27. C. Sakama and K. Inoue. Paraconsistent Stable Semantics for extended disjunctive programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.
28. 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.

29. 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.
30. 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.
31. G. Wagner. Ex contradictione nihil sequitur. In *International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1991.
32. 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.
33. G. Wagner. Web rules need two kinds of negation. In F. Bry, N. Henze, and J. Maluszynski, editors, *Principles and Practice of Semantic Web Reasoning Proc. of the 1st International Workshop, PPSWR'03*. Springer-Verlag, December 2003. LNCS 2901.