Logic Programs with Contextually Scoped Negation

Axel Polleres[1,2]

[1] Digital Enterprise Research Institute (DERI) Innsbruck, Austria
[2] Universidad Rey Juan Carlos, (Móstoles, Madrid), Spain
axel@polleres.net

Abstract. The Semantic Web community is currently dominated by knowledge representation formalisms adhering to a strict open world assumption. Nonmonotonic reasoning formalisms are viewed with partial scepticism and it is often argued that nonmonotonic reasoning techniques which adopt a closed world assumption are invalid in an open environment such as the Web where knowledge is notoriously incomplete. Nonetheless, in the ongoing discussion about rule extensions for Semantic Web Languages like RDF(S) or OWL several proposals have been made to partly break with this view and to allow a restricted form of negation as failure. Recently, the term "scoped negation" emerged in discussions around this topic, yet a clear definition about the meaning of "scope" and "scoped negation" and a formal semantics are still missing. In this paper we provide preliminary results towards these missing definitions and define two possible semantics for logic programs with contextually scoped negation, which we propose as an extension of RDFS.

1 Introduction

The current Web is a huge network linking between different sources of data and knowledge. This linked knowledge bases become particularly interesting when it comes to discussions about the next generation of the Web, the Semantic Web. Technologies like RDF/RDFS [5] and OWL [15] allow us to describe meta-data and the structure of such meta-data in an unambiguous way using standardized vocabulary, so-called ontologies. These ontologies let you infer additional knowledge about the meta-data published on the Web. However, the proper integration of ontology languages with existing rule and query languages is still an unresolved issue on W3C's agenda[3]. Particularly, nonmonotonic reasoning formalisms are viewed with partial scepticism[4] where it is argued that nonmonotonic reasoning is invalid in an open environment such as the Web where knowledge is notoriously incomplete. Still, two of the proposals for rule languages on the Web, namely WRL [1] and SWSL Rules [2], include negation as failure as language feature, however leaving critical questions about the suitability of negation as failure in a Web context open. Recently, the term "scoped negation" emerged in discussions around this topic, to describe a restricted form of negation as failure over a closed *scope* yet a clear definition about what "scope" and "scoped negation" mean and the formal semantics for this form of negation are still missing.

Building on the ideas of logic-programming based nonmonotonic rule languages we present a general framework for the combination of interlinked rule bases consisting of scoped and open rules involving scoped negation. We define desirable properties for scoped negation and discuss two possible semantics for programs with scoped negation.

The remainder of this paper is organized as follows: In Section 2 we define what we mean by context and scoped negation in the context of RDFS. Next, we introduce the syntax for rules which abstracts away from RDFS in Section 3. Based on this syntax, we define formal requirements for a proper semantics for rules using scoped negation. Two alternative semantics for programs with contextually scoped negation are proposed in Sections 3.1 and 3.2 which are defined in terms of a simple translation to normal logic programming under the stable model semantics, namely (a) contextually bounded negation semantics and (b) contextually closed negation semantics. These

[3] See the recently established Rule Interchange Format (RIF) working group `http://www.w3.org/2005/rules/`.
[4] See for instance `http://lists.w3.org/Archives/Public/public-sws-ig/2004Jan/0040.html`

translations provide support for direct implementation on top of existing engines. Finally, we discuss some related works, open issues and future work in Section 4.

2 Context and Scoped Negation

For tracking provenance of single fact or rule, we associate a *context* with each statement. We define a context as the URI of a Web-accessible data source (i.e. the location where a set of rules and facts is accessible. That means the context $< URI >$¿ is associated with all the rules and facts accessible when you type ¡URI in your browser. We denote this provenance of an RDF triple in our language by `triple(S,P,O)@`URI.

Note that the hash sign "#" does not have a special meaning here, i.e. the URIs `<http://www.polleres.net/foaf.rdf>` and `<http://www.polleres.net/foaf.rdf\#1>` denote exactly the same context in our definition, since they return the same RDF file (that means the same set of RDF facts).

Similarly, URI-prefixes have no special meaning with respect to the context, i.e. `<http://homepage.uibk.ac.at/>` does not necessarily include the union of all contexts prefixed by this URL, such as for instance `<http://homepage.uibk.ac.at/~c703262/foaf.rdf>` or `<http://homepage.uibk.ac.at/~c703261/foaf.rdf>`. In fact, `<http://homepage.uibk.ac.at/>`, denotes an empty graph, since there is no RDF/RDFS data accessible via this URI. Note that such prefix-compatibility might be implicit for proposed future Web protocols especially tailored for RDF querying such as `tsc://` [6], which are currently being investigated, but we make no such commitment in our general definition of context.

For our purposes, it is sufficient that by context we understand the rules and facts in a (finite set of) file(s) on the Web.

2.1 RDF(S) plus Rules

Our overall goal is to define a context-aware rule language with scoped negation on top of RDFS. Thus we assume arbitrary knowledge bases, consisting of RDF, RDFS and rules with scoped negation distributed over the Web at different URIs. To this end, we first introduce a straightforward LP-compliant notion of RDF and RDFS, thereafter we introduce the general rule language with scoped negation.

This paper is not about syntactical details of RDF or RDFS in XML. Rather, we use a simplified syntax of RDFS which is more leaned on logic programming to show that the major part of RDFS can be understood as a set of facts and rules in a logic program.

As implicit from [11], large parts of RDF and RDFS can be embedded in logic programming, by expressing each triple $\langle S, P, O\rangle$ by a predicate `triple(S,P,O)` and adding the following rules:

```
triple(P,rdf:type,rdf:Property) :- triple(S,P,O).
triple(S,rdf:type,rdfs:Resource) :- triple(S,P,O).
triple(O,rdf:type,rdfs:Resource) :- triple(S,P,O).
triple(S,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:domain,C).
triple(O,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:range,C).

triple(C rdfs:subClassOf rdfs:Resource) :- triple(C,rdf:type,rdfs:Class).
triple(C1,rdfs:subClassOf,C3) :- triple(C1,rdfs:subClassOf,C2),
                                 triple(C2,rdfs:subClassOf,C3).
triple(S,rdf:type,C2)         :- triple(S,rdf:type,C1),
                                 triple(C1,rdfs:subClassOf,C2).
triple(C,rdf:type,rdfs:Class) :- triple(S,rdf:type,C).
triple(C,rdfs:subClassOf,C)   :- triple(C,rdf:type,rdfs:Class).

triple(P1,rdfs:subPropertyOf,P3) :- triple(P1,rdfs:subPropertyOf,P2),
                                    triple(P2,rdfs:subPropertyOf,P3).
triple(S,P2,O)                   :- triple(S,P1,O),
```

```
                                    triple(P1,rdfs:subPropertyOf,P2).
 triple(P,rdfs:subPropertyOf,P)   :- triple(P,rdf:type,rdf:Property).
```

plus the respective the axiomatic triples in RDF/RDFS, cf. [11][Sections 3.1 and 4.1].[5]

Assume now that facts and class hierarchies describing movies and directors are given in several distributed sources on the Web:[6]

```
http://www.imdb.com/ :
 triple(m1,rdf:type,movie).
 triple(m1,hasDirector,"Ed Wood").
 triple(m1,hasTitle,"Plan 9 from Outer Space").
 ...
http://www.moviereviews.com/ :
 triple(m1,rate,bad).

http://www.b-movies.com/ :
 triple(m1,rate,classic).
 ...
```

However, query languages such as SPARQL [16] or rule languages such as N3 [3] have no defined means to express negative queries such as

"Give me movies which are not ranked as `bad` by `moviereviews.com` "

although I could properly answer such a query using a search engine which returns arbitrary movies other than `m1`. Note the difference to a query such as:

"Give me *all* movies which are not ranked as `bad`"

Obviously, no search engine in the world will be able to answer such a query; due to the notorious incompleteness of knowledge about published data on the Web one can neither find *all* movies nor determine whether *anybody* has ever ranked the movie as `bad`. Nonetheless, in usual search results we are happy to accept incomplete answers, as long as these are correct. Such answers can be provided for a query like the first one: Even if someone published a bad ranking for a particular movie somewhere else than `moviereviews.com`, this does not affect our result, since we negated only over a closed context, checkable by the triples available at a single, finite source, namely we apply negation as failure wrt. only rankings published at `moviereviews.com` in our example.

In the following, we will provide the formal basis for a rule and query language which allows to express such contextually scoped queries and which guarantees correct answers despite of incompleteness and the inherent nonmonotonicity of negation as failure by imposing that negation always needs to refer to a finitely closed scope.

3 Programs with Context and Scoped Negation

¿From here on, we no longer restrict ourselves to RDF and RDFS. Rather we give a general notion of logic programs (without function symbols) with contexts and scoped negation upon which we define a proper semantics fulfilling the above-mentioned informal requirement.

Definition 1. *If $t_1, \dots, t_n$ are constants or variables and c is a constant then $c(t_1, \dots, t_n)$ is an atom. A* scoped atom *is of the form $a@u$ where u is a URI and a is an atom.*[7] *A* literal *is either*

- *a (possibly scoped) atom* – positive literal
- *or a negated scoped atom of the form not $t@u$* – negative literal*,*

*i.e. all negative literals **must** be scoped.*

[5] For simplicity we ignore XML literals, datatypes, containers and blank nodes here. Additional issues related with these features of RDF are out of the scope of this paper.

[6] Indeed such RDF Data could easily be extracted from the respective Web sites by appropriate wrappers.

[7] Note that we do not allow variables or parametrized contexts such as for example in TRIPLE[8] or FLORA-2 [12].

Note that we do not make a distinction between constant symbols and predicate symbols, since the usage is clear from the syntax. Neither do the constants and URIs necessarily need to be disjoint.

Definition 2. *A program P is a set of rules of the form*

$$h :- l_1, \ldots, l_n.$$

Where h is an atom, and $l_1, \ldots l_n$ are literals and all variables occurring in h or in some negative body literal do also appear in a positive body literal. Each program P has a URI p and we make the assumption that each program can be accessed via its URI. The URI p is also called context *of P.*

The informal semantic meaning of scoped literals is that literals referenced via an external context represent links to other programs accessible over the Web.

Definition 3. *Let P, Q be programs with names p and q, respectively. We say that program P* links *to a program Q if P contains a scoped body literal (not) $a@q$ (direct link) or P contains a rule with a scoped body literal (not) $a@r$ such that the program R dereferenced by r links to Q. Given a set of Programs $\mathcal{P}$ we denote by the* closure *$Cl(\mathcal{P})$ the set of all programs in $\mathcal{P}$ plus all programs which are linked to programs in $\mathcal{P}$.*

Definition 4. *A rule is* contextually bounded *iff each negative body literal not $a@p$ is contextually bounded.*

A scoped literal (not) $a@p$ is called contextually bounded, iff each each rule r in the program dereferenced by name p with head h where h is unifiable with a is strongly contextually bounded.

A rule is strongly contextually bounded *iff it has either an empty body or each body literal is scoped and contextually bounded.*

A program is (strongly) contextually bounded if each of its rules is (strongly) contextually bounded.

Intuitively, contextual boundedness means that each negative literal is recursively only depending on scoped literals. From our above definition, we see that we can separate each program into its "open" and "closed" part:

Definition 5. *Let P be a program, then we denote by $\lfloor P \rfloor$ the program consisting only of the strongly contextually bounded rules in P and by $\lceil P \rceil$ the program consisting only of not strongly contextually bounded rules.*

As we can see, $\lfloor P \rfloor$ denotes the set of rules which is based only on a set of rules closed over explicitly given contexts, whereas $\lceil P \rceil$ defines all "open" rules in P. This means that $\lfloor P \rfloor$ is independent of the query context, whereas $\lceil P \rceil$ is not.

Next, we define queries over a set of (known) programs $\mathcal{P}$ and informally motivate some intuitive requirements on query answers which we would expect from the proper semantics for query answers:

Definition 6. *We denote by $AS_{\mathcal{S}}(\mathcal{P})$ the set of consequences from a set of programs $\mathcal{P} = \{p_1, \ldots, p_k\}$ wrt. semantics $\mathcal{S}$. A* query *q is of the form:*

$$l_1, \ldots, l_n?\{p_1, \ldots, p_k\}$$

where $l_1, \ldots l_n$ are literals and all variables occurring in some negative literal do also appear in a positive literal, i.e. we adhere to the usual safety restriction also for queries.

A query answer is an assignment of constants to all the variables in q, such that all positive literals in $l_1, \ldots l_n$ are in $AS_{\mathcal{S}}(\{p_1, \ldots, p_k\})$ and all negative literals are not in $AS_{\mathcal{S}}(\{p_1, \ldots, p_k\})$. The set of all valid query answers wrt. semantics $\mathcal{S}$ is denoted by $AS_{\mathcal{S}}(q)$.

Let us consider the scenario from the Introduction where you ask a query q to a search engine which spiders arbitrary programs describing movies on the Web. Here, the set $\{p_1, \ldots, p_k\}$ (which we call the *query context*) is unknown to the user and only known to the search engine. Although, the search engine might gather tremendous amounts of URIs, i.e. programs, in an open environment such as the Web, you can never be sure to have complete knowledge. We would expect the following intuitive requirements fulfilled by our semantics:

R1 Whatever query you ask, the results should return a maximum set of answers which are entailed by the semantics with respect to the known programs. Still the semantics we choose should guarantee, that in case additional knowledge bases (i.e. programs) becomes available to the search engine, none of the previous answers need to be retracted. Thus, we require that our semantics is monotonic with respect to the addition of knowledge bases, i.e.

$$\mathcal{P} \subseteq \mathcal{R} \Rightarrow AS_{\mathcal{S}}(\mathcal{P}) \subseteq AS_{\mathcal{S}}(\mathcal{R})$$

i.e. $\mathcal{P} \subseteq \mathcal{R} \Rightarrow AS_{\mathcal{S}}(l_1, \ldots, l_n?\mathcal{P}) \subseteq AS_{\mathcal{S}}(l_1, \ldots, l_n?\mathcal{R})$, respectively. We will further refer to this requirement as *context-monotonicity.* Note that context-monotonicity can be viewed as soundness of query answers. Although completeness can never be achieved, due to openness of the environment, at least we want to be sure that our semantics only returns correct answers.[8] This requirement is particularly important the query context is unknown to/uncontrollable by the user $\{p_1, \ldots, p_k\}$ such as in the above-mentioned search engine scenario.

R2 We expect that the semantics is *closed under context closure*, i.e.

$$AS_{\mathcal{S}}(\mathcal{P}) = AS_{\mathcal{S}}(Cl(\mathcal{P}))$$

That means we would expect that the semantics considers not only the query context, but also all linked programs which the programs in the query context refer to. As we will see, this requirement is arguable and we will drop it for the second of two semantics we will define later in this paper.

R3 Next, we would intuitively expect, that we can modularize and distribute evaluation of queries by interlinked clusters of programs. Particularly, we suggest the following intuitive "compositionality requirements" for programs on the Web:

1. Assume the two sets of programs $\mathcal{P}$ and $\mathcal{R}$ are not mutually linked, then

$$AS_{\mathcal{S}}(\mathcal{P} \cup \mathcal{R}) = AS_{\mathcal{S}}(\lceil\mathcal{P}\rceil \cup \lceil\mathcal{R}\rceil \cup AS_{\mathcal{S}}(\lfloor\mathcal{P}\rfloor) \cup AS_{\mathcal{S}}(\lfloor\mathcal{R}\rfloor))$$

   Particularly if the two sets of programs $\mathcal{P}$ and $\mathcal{R}$ are strongly contextually bounded and not mutually linked, then we expect

$$AS_{\mathcal{S}}(\mathcal{P} \cup \mathcal{R}) = AS_{\mathcal{S}}(\mathcal{P}) \cup AS_{\mathcal{S}}(\mathcal{R})$$

2. Assume that the sets of programs $\mathcal{P}$ links to the set of programs $\mathcal{R}$ but not vice versa and $\mathcal{R}$ is (i) either strongly contextually bounded or (ii) does not use any unscoped body literals which unify with heads in $\mathcal{R}$, then

$$AS_{\mathcal{S}}(\mathcal{P} \cup \mathcal{R}) = AS_{\mathcal{S}}(\mathcal{P} \cup AS_{\mathcal{S}}(\mathcal{R}))$$

3.1 Contextually Bounded Semantics

In the following, we define the semantics of programs under the assumption that all programs are contextually bounded, i.e. that no program negatively references to a non strongly contextually bounded rule. We a possible semantics under this assumption based on the stable semantics for logic programs.

For a contextually bounded program p we define a rewriting $tr_{CB}(p)$ by replacing each rule `h :- l1, ..., ln.` with the pair of rules

`h :- h@p. h@p :- l1, ..., ln.`

Let $\mathcal{P}$ be a set of programs. Then we define $AS_{CB}(\mathcal{P})$ as follows:
Let $\mathcal{P}_{CB} = \bigcup_{p \in Cl(\{p_1,\ldots,p_k\})} tr_{CB}(p)$, then

$$AS_{CB}(\mathcal{P}) = \bigcap \mathcal{M}(\mathcal{P}_{CB})$$

where $\mathcal{M}(P)$ is defined as the set of all stable models[9] of program P, i.e. we define $AS_{CB}(\mathcal{P})$ by the cautious consequences of $\mathcal{P}_{CB}$ under the stable model semantics.

We will now investigate the two above defined semantics with respect to requirements R1–R3 defined above.

Proposition 1. *Context-monotonicity (R1) holds for AS_{CB} under the assumption that all programs are contextually bounded and $Cl(\mathcal{P})$ is consistent [9] for any set of programs $\mathcal{P}$.*

[8] Note that obviously, this only holds under the somewhat idealized assumption that in the "Web of programs" only trustworthy knowledge providers publish their knowledge.

[9] By consistency we mean, as usual, the existence of a stable model.

Proof. (sketch) Let us assume that context-monotonicity does not hold, i.e. there exist programs p, r such that $AS_{CB}(p) \not\subseteq AS_{CB}(\{p, r\})$. ¿From this, we conclude that there exists some atom a in $\mathcal{M}(p_{CB})$ which is not in $M(p_{CB} \cup r_{CB})$. By the working of the answer set semantics and the assumption that $Cl(\{r\})$ is consistent, we know that this can only be the case, because a indirectly depends negatively on some literal in r_{CB}. However, due to the fact that each negation is scoped and the contextual boundedness assumption, this would imply that there exists some rule of the form $b@r_i : -body$ in r_{CB} which is satisfied in all stable models of $p_{CB} \cup r_{CB}$ but not in p_{CB} and which stems from a strongly contextually bounded rule $b : -body$ in program r_i. However, since then r_i would necessarily be in $Cl(p)$ and thus $b@r_i : -body$ in p_{CB} we get a contradiction, because therefore also *body* solely depends on rules in p_{CB}.

A simple counterexample shows that contextual boundedness is indeed required for context-monotonicity to hold:

p:	r:
`a :- not b@p.`	`c.`
`b :- c.`	

Requirement R2 holds by definition, since $AS_{CB}(\mathcal{P})$ is defined in terms of the closure of $\mathcal{P}$. Since this paper only reflects a preliminary state of investigation we leave concrete evaluations and proofs for requirement R3 for future work, where we will particularly focus on implementation issues and aspects of distributed query evaluation under scoped negation and where R3 is of particular importance.

The restriction to contextually bounded programs is justified in an open environment only under the assumption that existing programs once published under a certain URI do not change and all previously published programs always remain accessible. Under this assumption, publishing new programs always needs to be preceded by a check for contextual boundedness. Note that the condition of contextual boundedness was defined with this assumption in mind: Publishing/adding new programs should not ever break context-monotonicity. However, obviously contextual boundedness is not stable against changes of single programs as the following example shows:

p:	r:
`a.`	`b :- not a@p.`

Here, `p` is obviously contextually bounded. Upon publication of `r` contextual boundedness could be checked and granted with respect to `p`. However, if one later on changes `p` by adding the single "open" rule `a :- c.` contextual boundedness of `r` would be broken. Unfortunately, such violations can not be checked upon change of a program, since in an open and de-coupled environment `p` would not be aware of which other programs link to it.

Thus, in the next section we try to define an alternative semantics which is more restrictive in the sense that it allows to infer less consequences, but is more stable against changes, since it is independent of contextual boundedness.

3.2 Contextually Closed Semantics

Let p be an arbitrary program, then we define the program $tr_{CC}(p)$ by rewriting each rule $h : -l_1, \ldots, l_n.$ in p to $h@p : -l'_1, \ldots, l'_n.$ where

$$l'_i = \begin{cases} l_i & \text{in case } l_i \text{ is scoped} \\ l_i@p & \text{otherwise} \end{cases}$$

Intuitively, under this semantics the semantics of scoped literals changes to "*not a@p* is true if and only if a cannot be derived from p *alone*": Even if there is a rule with a in its head in p which is not strongly contextually bounded (i.e. it has an open literal in its body) then only the rules and facts in p alone are taken into account for $a@p$ under this semantics.

Let $\mathcal{P}$ be a set of programs. Then we define $AS_{CC}(\mathcal{P})$ as follows: Let $\mathcal{P}_{CC} = \bigcup_{p \in Cl(\{p_1,\ldots,p_k\})} tr_{CC}(p) \cup p_1 \cup \ldots \cup p_k$, then

$$AS_{CC}(\mathcal{P}) = \bigcap \mathcal{M}(\mathcal{P}_{CC})$$

where $\mathcal{M}(P)$ is defined as the set of all stable models of program P.

Context-monotonicity (R1) is trivially fulfilled under this semantics, since negation is automatically "closed off" to the linked context only. Note that, in case all programs are contextually bounded the semantics still does not coincide, but contextually closed semantics is indeed more restrictive than contextually bounded semantics:

Proposition 2. *For any set of contextually bounded programs $\mathcal{P}$*

$$AS_{CC}(\mathcal{P}) \subseteq AS_{CB}(\mathcal{P})$$

Proof. (sketch) We restrict our observations to the stable semantics version of the two semantics, i.e. we want to show that: $AS^{sms}_{CC}(\mathcal{P}) \subseteq AS^{sms}_{CB}(\mathcal{P})$ Note that by construction for each rule $h : -l_1, \ldots, l_n.$ in $\lfloor Cl(\mathcal{P}) \rfloor$ stemming from program p there is a rule $h@p : -l_1, \ldots, l_n.$ in $\mathcal{P}_{CB} \cap \mathcal{P}_{CC}$. We denote this set of rules by $floor$. Obviously, $Lit(floor)$ is a splitting[14] for both $\mathcal{P}_{CB}$ and $\mathcal{P}_{CC}$ and thus the answer sets for both coincide on $floor$. Moreover, $\mathcal{P}_{CB} \setminus floor$ and $\mathcal{P}_{CC} \setminus floor$ are both positive logic programs modulo input negation for literals from $floor$. A bit sloppy, now for each rule $h : -l_1, \ldots l_n.$ which is satisfied in $\mathcal{P}_{CC} \setminus floor$ also a pair of rules $h : -h@p.$ $h : -l_1, \ldots l_n.$ is satisfied in $\mathcal{P}_{CB} \setminus floor$, proving that the consequences of $\mathcal{P}_{CB}$ are a superset of $\mathcal{P}_{CC}$.

Indeed, there are consequences under contextually bounded semantics which are not valid under contextually closed semantics, as shown by the following example:

p:	r:
`a :- b@r.`	`c.`

Here, the query $c?\{p\}$ is true with respect to contextually bounded semantics but not with respect to contextually closed semantics. The example shows also that R2 does not hold for contextually closed semantics. This reflects the intuition that contextually closed semantics draws inferences more cautiously, but does in trade not run into problems with contextually unbounded programs and is thus stable against program changes.

4 Related Works and Outlook to Future Work

FLORA-2 [12] is a query answering system based on the logic programming fragment of F-Logic [13], a frame-based syntactic variant of first-order logic which is gaining popularity for ontology reasoning. FLORA-2 supports a limited form of scoped negation via so-called Modules. By default, all literals in FLORA-2 are scoped, note however, that also variables are allowed in place of modules in FLORA-2 thus not imposing strict scoping of negated literals or contextual boundedness. Anyway, FLORA-2 is a system and per se does not define the semantics of programs and queries defined on the Web, nor are any assumptions made that modules need to coincide with contexts (i.e. URIs) in our sense or guarantees for context-monotonicity given.

TRIPLE [8] is another logic programming engine particularly tailored for RDF reasoning and querying with the support of possibly parametrized contexts. Parametrized contexts are an expressive feature beyond our very simple and practical definition of contexts as simple URIs, but we emphasize that our definition of context was more focused on simplicity and usability in a Web context than driven by expressive power.

Both FLORA-2 and TRIPLE treat negation under the well-founded semantics, which slightly differs from the semantics defined in this paper. However, we plan to investigate variants of our semantics based on well-founded semantics as part of future work.

Another rule and query engine particularly for Web languages is XCerpt [17]. XCerpt has particular features tailored for querying tree structures such as XML documents and thus supports scoped literals in the sense that terms scope over particular XML documents. Recursion and the use of negation as failure, however, are limited to stratified programs, and there are no "open" rules in our sense, but all references to contexts are to be made explicit.

C-OWL [4] is a proposed extension of the Web Ontology language OWL by contexts and bridge rules. We see some parallels between our context-dependent rules and bridge rules which are under investigation. Particularly, extensions or combinations of OWL with Logic Programs applied to the contextualized versions, C-OWL and contextually bounded logic programs are on our agenda.

In this paper we discussed logic programs under contextually scoped negation and provided two possible semantics based on simple translations to normal logic programs under the stable models semantics.

This work is a first step towards a proper definition for scoped negation as a proposal for the upcoming RIF working group in W3C. There are many issues left open and indicating further research directions. First, it seems to be useful to extend our definition of scope to unions (and maybe intersections) of contexts. More refined concepts of context such as e.g. so-called named RDF graphs [7] could serve as a basis for further investigations.

We set the basis for a rule based query language. It is well-known that query languages like SQL naturally translate into queries expressed by logic programs. However, without negation as failure (for modeling set difference) such a query language is incomplete. Scoped negation is a natural and lightweight candidate to extend negation-free RDF query languages such as for instance SPARQL [16] and N3 [3].

Implementation issues for distributed query evaluation have not yet been discussed. We consider to investigate and implement queries with scoped negation in a distributed variant of the RDF store YARS [10].

Acknowledgments The author thanks Cristina Feier, Andreas Harth, and Jos De Bruijn for fruitful discussions. This work is funded by the European Commission under the projects DIP, KnowledgeWeb, InfraWebs, SEKT, and ASG; by FIT-IT under the projects RW^2 and TSC.

References

1. J. Angele, H. Boley, J. de Bruijn, D. Fensel, P. Hitzler, M. Kifer, R. Krummenacher, H. Lausen, A. Polleres, and R. Studer. Web Rule Language (WRL). W3C Member Submission, available from `http://www.w3.org/Submission/2005/08/`, June 2005.
2. S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, D. L. McGuinness, S. McIlraith, G. Newton, D. de Roure, M. Skall, J. Su, S. Tabet, and H. Yoshida. Semantic Web Services Framework (SWSF). W3C Member Submission, available from http://www.w3.org/Submission/2005/07/, May 2005.
3. T. Berners-Lee, D. Connolly, E. Prud'homeaux, and Y. Scharf. Experience with N3 Rules. In *W3C Workshop on Rule Languages for Interoperability*, Washington, D.C., USA, April 2005.
4. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In *The SemanticWeb - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003.
5. D. Brickley, R. V. Guha (editor), and B. McBride (eds.). RDF Vocabulary Description Language 1.0. URL, February 10 2004. W3C Recommendation, `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.
6. C. Bussler. A Minimal Triple Space Computing Architecture. In *2nd WSMO Implementation Workshop (WIW)*, Innsbruck, Austria, 2005.
7. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named Graphs. *Journal of Web Semantics*, 3(4), 2005.
8. S. Decker et al. TRIPLE - An RDF Rule Language with Context and Use Cases. In *W3C Workshop on Rule Languages for Interoperability*, Washington, D.C., USA, April 2005.
9. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and Kenneth Bowen, editors, *5th Int'l Conf. on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
10. A. Harth and S. Decker. Optimized Index Structures for Querying RDF from the Web. In *3rd Latin American Web Congress*, Buenos Aires, Argentina, October 2005.
11. P. Hayes. RDF Semantics. Technical report, W3C, 2004. W3C Recommendation 10 February 2004. Available from http://www.w3.org/TR/rdf-mt/.
12. M. Kifer. Nonmonotonic Reasoning in FLORA-2. In *8th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, Diamante, Italy, 2005. Invited Paper.
13. M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *JACM*, 42(4):741–843, 1995.
14. V. Lifschitz and H. Turner. Splitting a Logic Program. In Pascal Van Hentenryck, editor, *11th Int'l Conf. on Logic Programming (ICLP'94)*, pages 23–37, Santa Margherita Ligure, Italy, June 1994. MIT Press.
15. D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. `http://www.w3.org/TR/2003/WD-owl-features-20030331/`, March 2003.
16. E. Prud'hommeaux and A. Seaborne (eds.). SPARQL Query Language for RDF, July 2005. W3C Working Draft.
17. S. Schaffert Xcerpt: A Rule-Based Query and Transformation Language for the Web PhD thesis, 2004.