Rules with Contextually Scoped Negation

Axel Polleres[1,2], Cristina Feier[1], and Andreas Harth[1]

[1] Digital Enterprise Research Institute Innsbruck, Austria and Galway, Ireland
[2] Universidad Rey Juan Carlos, Madrid, Spain
axel@polleres.net, {cristina.feier, andreas.harth}@deri.org

Abstract. Knowledge representation formalisms used on the Semantic Web adhere to a strict open world assumption. Therefore, nonmonotonic reasoning techniques are often viewed with scepticism. Especially negation as failure, which intuitively adopts a closed world view, is often claimed to be unsuitable for the Web where knowledge is notoriously incomplete. Nonetheless, it was suggested in the ongoing discussions around rules extensions for languages like RDF(S) or OWL to allow at least restricted forms of negation as failure, as long as negation has an explicitly defined, finite scope. Yet clear definitions of such "scoped negation" as well as formal semantics thereof are missing. We propose logic programs with *contexts* and *scoped negation* and discuss two possible semantics with desirable properties. We also argue that this class of logic programs can be viewed as a rule extension to a subset of RDF(S).

1 Introduction

The current Web is a huge network linking between different sources of data and knowledge, formatted for human users. Such linked knowledge bases become particularly interesting when it comes to discussions about the next generation of the Web, the Semantic Web. Technologies like RDF(S) [5] and OWL [16] shall allow us to describe meta-data and the structure of such meta-data in an unambiguous way using standardized vocabularies, also called ontologies. These ontologies let you infer additional knowledge about the meta-data published on the Web. Meta-data descriptions and ontologies are to be distributed over the Web just like current Web pages as machine-readable knowledge bases accessible via URIs. Different approaches exist for combining such meta-data from different sources. A common approach is to import and/or simply reuse the vocabulary of one ontology in the definition of another, for instance using common namespaces or OWL's import mechanism. A more fine-grained approach is in the form of so-called mappings or bridge rules [4] that connect entities from different knowledge bases. Eventually, standardized rule languages, which allow for the definition of such mappings or other combinations of meta-data in general are the natural next evolution step on W3C's agenda. Still, there are many unresolved issues around the proper integration of ontology language recommendations such as RDFS and OWL with existing rule languages. For instance, nonmonotonic features of such rule languages are viewed with partial scepticism[1]. In particular,

[1] cf. http://lists.w3.org/Archives/Public/public-sws-ig/2004Jan/0040.html

Y. Sure and J. Domingue (Eds.): ESWC 2006, LNCS 4011, pp. 332–347, 2006.

it is argued that the use of negation as failure 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 a 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*. "Scoped negation as failure" is also explicitly mentioned as one of the extensions to be investigated by W3C's recently established Rule Interchange Format (RIF) working group[2]. However, clear definitions of what "scope" and "scoped negation" actually mean and what the formal semantics for this form of negation should be are missing.

Contributions. In this paper we present a logic programming framework for the combination of interlinked rule bases on the Web and show how scoped negation as failure fits in such a framework. A peculiarity of our rule language is that it allows "open" as well as "closed" rules: On the one hand, universally valid, open rules shall be allowed which apply to any available statement on the Web. This is in accordance with RDF and OWL which also allow that several sources define statements and axioms affecting the same resource.[3] On the other hand, we also define closed rules which are only evaluated with respect to a particular context, that is a (finite and known set of) web-accessible rule base(s).

We ensure in our language that negation as failure is always "scoped", i.e. that the search for failure in a rule body is not depending on any "open" rules. This way we circumvent the undesirable non-monotonic effects of negation as failure in open environments such as the Web. Thereby we achieve a weak form of monotonicity, called *"context-monotonicity"* which intuitively means that negation as failure behaves monotonically with respect to the set of web-accessible rule-bases that an agent is aware of. In order to achieve context-monotonicity we propose two alternative semantics for sets of rule bases with scoped negation, namely (a) contextually bounded semantics and (b) contextually closed semantics. Both semantics are defined in terms of translations to normal logic programs. Remarkably, these translations make no commitment to a particular semantics used for negation as failure upfront, be it well-founded or stable, and allow for direct implementations on top of many existing rule engines which adopt either of these semantics.

We further demonstrate that our language can be viewed as a rule extension of (a subset of) RDFS.

Paper Overview. The remainder of this paper is organized as follows: In section 2 we illustrate by means of simple examples what we understand by context, "open" and "closed" rules, and queries. We formally introduce the syntax for logic programs with contexts and scoped literals in section 3.1. We then define a formal requirement for a proper semantics for such programs called context-monotonicity. The two alternative semantics fulfilling this requirement

[2] cf. `http://www.w3.org/2005/rules/wg/charter`.

[3] Actually, a strong argument why semantics of these languages assume an open world.

are presented in sections 3.2 and 3.3. We relate our approach to RDF(S) in section 4 and slightly extend our notion of scope to unions of contexts in section 5. Finally, we discuss some related works and draw conclusions in sections 6 and 7.

2 Context, Open Rules and Scoped Negation

In the following, we will give an informal description of our notion of context, logic programs with open vs. closed rules and introduce our understanding of scoped negation as failure for such programs. We will base these explanations on simple examples. The underlying formal definitions are given in section 3.1.

Context. For tracking provenance of a 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 retrieved when you type *URI* in your browser.

Rules and Queries. In the context of our rule language, we assume a Web of logic programs, published at different URIs. In order to illustrate this, we assume programs describing movies, directors and ratings as shown in Figure 1. The notation we use is the usual syntax known from e.g. PROLOG systems.

http://www.moviereviews.com/
```
rated(m1,bad).
rated(X,bad) :- directedBy(X,"Ed Wood").   (*)
```

http://www.b-movies.com/
```
rated(m1,classic).
rated(m3,classic).
```

http://www.polleres.net/
```
rated(m2,bad). movie(m2).
```

http://www.imdb.com/
```
sciFiMovie(m1). hasTitle(m1,"Plan 9 from Outer Space"). directedBy(m1,"Ed Wood").
sciFiMovie(m2). hasTitle(m2,"Matrix Revolutions"). directedBy(m2,"Andy Wachowski").
directedBy(m2,"Larry Wachowski").
sciFiMovie(m3). hasTitle(m3,"Bride of the Monster"). directedBy(m3,"Ed Wood").
movie(X) :- sciFiMovie(X).
...
```

Fig. 1. Four Programs describing rules and data about movies, directors and ratings

A typical feature which we adopt from RDF is that different sources (contexts) are allowed to talk about the same resource. This shall allow to draw additional conclusions from combining contexts. For instance, in our example, three contexts `http://www.imdb.com/`, `http://www.moviereviews.com/` and `http://www.b-movies.com/` talk about the same movie `m1`. A semantic search engine might gather arbitrary programs like the ones shown in Figure 1 on the Web from different contexts and allow us to ask queries about particular movies. Queries can be formalized as rules, e.g.

"Give me `movie`s which are rated as `bad`" (1)

can be expressed by the following simple rule:

```
answer(X) :- movie(X), rated(X,bad).
```

We call a rule like this "open" since it is not restricted to a particular context, but in principle all movies and ratings at all possible contexts on the Web are of interest.

Assume that the search engine, which has to evaluate this query is aware of the contexts `http://www.imdb.com/`, `http://www.moviereviews.com/`, `http://www.b-movies.com/`, where we would expect `m1` and `m3` as answers. The easiest and straightforward way to evaluate such a query then would be to retrieve these three programs, build the union of all rules and facts and and then evaluate the resulting logic program with one of the standard techniques. Note that `rated(m3,bad)` is inferred by another "open" rule from the combination of two contexts.

Usually, in such a Web search scenario we would accept incompleteness of the given answers, since we cannot expect that our fictitious search engine has complete knowledge about the whole Web. For instance, the search engine might not be aware of the personal movie reviews of one of the authors published at `http://www.polleres.net/`, see Figure 1 and would thus not return `m2` as an answer for query (1). But at least we can be sure that all answers are sound, as long as programs consist of positive rules only.

Scoped Literals. Recall the "open" rule (*) in `http://www.moviereviews.com/` saying that everything directed by Ed Wood is bad. If we want to determine the provenance of a certain atom (i.e., "to which context does a particular fact belong to") this is easy for facts such as `rated(m1,bad)`. However, we have certain difficulties to determine the provenance of atoms inferred by (open) rules via information from other contexts. For instance, does the inferred atom `rated(m3,bad)` "belong" to context `http://www.moviereviews.com/` or context `http://www.imdb.com/` (which was needed to satisfy the body of the rule)? In this paper, we will adopt the view that all facts inferred by rules belong to the context of the rule that triggered the new fact.[4]

Now that we have given an informal definition of provenance of atoms inferred from distributed logic programs, we can ask queries about facts restricted to a certain context, like for instance:

> "Give me `movies` which are rated as `bad` by `http://www.moviereviews.com/`" (2)

We will use the following notation for such a query/rule in this paper:

```
answer(X) :- movie(X), rated(X,bad)@http://www.moviereviews.com/.
```

where we call the atom `rated(bad)@http://www.moviereviews.com/` a *scoped* literal. By making the context explicit, we do no longer need to bother about information concerning ratings from other sources such as the ones from `http://www.polleres.net/`.

However, we still have not solved the problem about incomplete information here since the atom `rated(bad)@http://www.moviereviews.com/` again

[4] Note that an atom can belong to several contexts. We remark that there are more involved proposals for handling provenance of data on the Semantic Web, see e.g. [7].

depends on an open rule; i.e., as soon as the search engine would become aware of an additional source saying that a particular other movie was directed by Ed Wood it could again infer additional information from the rule (*) in Figure 1. This problem could be solved by making rule (*) more explicit. If we know that IMDB has complete knowledge about all movies by Ed Wood we could replace the rule (*) by its closed off version, adding a scope just as we did for query (2):

```
rated(X,bad) :- directedBy(X,"Ed Wood")@http://www.imdb.com.        (**)
```

Now, under the assumption that `http://www.imdb.com` stores all `directedBy(·)` atoms as explicit facts not depending on any other (open) rules, we can indeed be sure to get *complete* information about the ratings for query (2).

The example shows that one has to be aware that scoped literals do not solve the problem of incompleteness per se; completeness is only achievable if none of rules which the scoped literal depends on contains open literals. As it turns out, this issue becomes more severe in combination with negation as failure.

Scoped Negation as Failure. Let us now focus on negative queries such as:

"Give me movies which are not ranked as bad" (3)

Such queries can be expressed in a rule language with negation as failure (`not`):

```
answer(X) :- movie(X), not rated(X,bad).
```

However, here we end up in a dilemma due to the inherent non-monotonicity of negation as failure: Unless we have complete information about *any* rating ever published, we can never be sure to find correct answers to such a query.

What we aim at in this paper is a more cautious form of negation, i.e. *scoped negation as failure*, which allows us to ask negative queries with explicit scope, such as:

"Give me movies which are not ranked as bad by `moviereviews.com` " (4)

Now, if the ratings on `moviereviews.com` solely depend on facts and "closed" rules such as (**), we can safely return correct answer. We will give a formal definition of this condition, which we call *contextual boundedness*, in section 3.1. For instance, contextual boundedness is violated by the combination of queries such as (4) and open rules such as (*).

Contextually bounded use of scoped negation as failure intuitively guarantees that even if we become aware of new contexts with additional information, we do not need to retract any query answers. We will call this desirable property *context-monotonicity* in the following.

3 Programs with Context and Scoped Negation

In this section, we provide the formal basis for the rule and query language informally introduced in section 2. We will allow to express contextually scoped queries and rules in a way that guarantees sound answers despite of incomplete

knowledge and the inherent non-monotonicity of negation as failure. We propose two approaches to achieve this, either (a) we syntactically guarantee contextually bounded use of negation, or (b) we close off open rules referenced by scoped literals. We will define semantics for both these options by means of appropriate transformations to normal logic programs which then can be evaluated using one of the standard semantics for negation as failure.

3.1 Definitions

Definition 1 (Scoped Atoms, Literals). *If $t_1, \ldots, t_n$ are constants or variables and c is a constant then $c(t_1, \ldots, t_n)$ is an atom. A* scoped atom *is of the form $a@u$ where u is a URI and a is an atom.*[5] *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.*

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 (Program). *A program P is a set of rules of the form*

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

Where h is an unscoped 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 (Link, Closure). *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 (Contextual Boundedness). *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 rule r in the program dereferenced by name p with head h where h is unifiable with a is strongly contextually bounded.

[5] Note that we do not allow variables or parameterized contexts such as for example in TRIPLE[8] or FLORA-2 [13].

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 a 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" parts:

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

Intuitively, $\lfloor P \rfloor$ denotes a 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 "self-contained", i.e., independent of the contexts which the agent (in our example the search engine) is aware of, whereas $\lceil P \rceil$ is not.

Next, let us define queries before we describe an intuitive requirement which we would expect from the proper semantics for respective query answers:

Definition 6 (Query, Query Answer). *We denote by $Cn_{\mathcal{S}}(\mathcal{P})$ the set of consequences from a set of programs $\mathcal{P}$ wrt. semantics $\mathcal{S}$. A* query *q is a special context consisting of a single rule:*

$$\texttt{answer}(x_1,\ldots,x_n)\ \texttt{:-}\ l_1,\ \ldots,\ l_k.$$

where $x_1, \ldots x_n$ are all variables and `answer` *is a special predicate symbol not allowed in other contexts. We define a* query answer *wrt. an agent A as a tuple of constants $(c_1, \ldots c_n)$, such that* $\texttt{answer}(c_1,\ldots,c_n) \in Cn_{\mathcal{S}}(\mathcal{P}_A \cup q)$*, where $\mathcal{P}_A$ denotes the set of contexts which A is aware of.*

Context-Monotonicity. Let us consider we ask a query q to a search engine A. Here, the set $\mathcal{P}_A$ is unknown to the user and only known to A. Although A might gather tremendous amounts of URIs, i.e. programs, in an open environment such as the Web, one can never be sure that A has complete knowledge. We would expect the following intuitive requirement fulfilled by our semantics:

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

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

where $\mathcal{P}, \mathcal{R}$ are sets of contexts. 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 sound answers.[6] The claim for context-monotonicity may be viewed as contradictory to the inherent non-monotonicity of negation as failure. However, the intention here is as follows: Negation as failure is allowed, as long as it is restricted to an explicit scope. This view corresponds to the usual use of negation in a database sense where a closed world assumption is made rather than meaning negation "by default". I.e., we allow a closed world view as long as it is explicit where to close off.

3.2 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 contextually unbounded atom. The semantics is defined in terms of a simple rewriting in two variants, based on the stable and well-founded semantics for logic programs, respectively. As it turns out, context-monotonicity is guaranteed for both variants.

For a contextually bounded program p we define a rewriting $tr_{CB}(p)$ by replacing each rule h `:-` l_1`,` `...,` l_n`.` with the rule $h@p$ `:-` l_1`,` `...,` l_n`.`

Definition 7 (Contextually Bounded Consequences). *Let $\mathcal{P} = \{p_1, \ldots, p_k\}$ be a set of programs. Then we define $Cn_{CB}(\mathcal{P})$ as follows:*
Let $\mathcal{P}_{CB} = \bigcup_{p \in Cl(\mathcal{P})} tr_{CB}(p) \cup p_1 \cup \ldots \cup p_k$, then

- *$Cn^{sms}_{CB}(\mathcal{P}) = \bigcap \mathcal{M}(\mathcal{P}_{CB})$ where $\mathcal{M}(\Pi)$ denotes the set of all stable models[11] of a program Π, i.e. we define $Cn^{sms}_{CB}(\mathcal{P})$ by the cautious consequences of $\mathcal{P}_{CB}$ under the stable model semantics.*
- *$Cn^{wfs}_{CB}(\mathcal{P}) = M(\mathcal{P}_{CB})$ where $M(\Pi)$ denotes the well-founded model [10] of program Π.*

We now investigate the two semantic variants wrt. context-monotonicity:

Proposition 1. *Context-monotonicity holds for Cn^{sms}_{CB} under the assumption that all programs are contextually bounded for any set of programs $\mathcal{P}$.*

Proof. (sketch) Let us assume that context-monotonicity does not hold, i.e. there exist programs p, r such that $Cn_{CB}(p) \not\subseteq Cn_{CB}(\{p, r\})$. 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 stable model semantics we know that this can only be the case if a 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}.

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

By similar arguments we can show:

Proposition 2. *Context-monotonicity holds for Cn_{CB}^{wfs} under the assumption that all programs are contextually bounded.*

A simple counterexample shows that context-monotonicity no longer holds when the requirement for contextual boundedness is dropped:

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

Here, the rewriting $\{p\}_{CB}$ yields:

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

which obviously has $\mathtt{a} \in Cn_{CB}^{sms,wfs}(p)$, whereas $\{p, r\}_{CB}$ would extend the above program by the facts `c.` and `c@r.` such that $\mathtt{a} \notin Cn_{CB}^{sms,wfs}(\{p, r\})$.
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 decoupled environment p would not be aware of which other programs link to it.

Thus, in the next section we try to define an alternative rewriting which is more restrictive in the sense that it allows to infer less consequences under both the stable or well-founded semantics, but is more resistant against program changes, since it is independent of contextual boundedness.

3.3 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 can(not) be derived from p *alone*".

Definition 8 (Contextually Closed Consequences). *Let $\mathcal{P} = \{p_1, \ldots, p_k\}$ be a set of programs. Then we define $Cn_{CC}(\mathcal{P})$ as follows:*
Let $\mathcal{P}_{CC} = \bigcup_{p \in Cl(\mathcal{P})} tr_{CC}(p) \cup p_1 \cup \ldots \cup p_k$, then

– $Cn^{sms}_{CC}(\mathcal{P}) = \bigcap \mathcal{M}(\mathcal{P}_{CC})$ *where* $\mathcal{M}(\Pi)$ *is the set of all stable models of* Π.
– $Cn^{wfs}_{CC}(\mathcal{P}) = M(\mathcal{P}_{CC})$ *where* $M(\Pi)$ *is the well-founded model of* Π.

Context-monotonicity is trivially fulfilled under this translation both in the stable and well-founded variants, since negation is automatically "closed off" to the linked contexts 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 3. *For any set of contextually bounded programs* $\mathcal{P}$

$$Cn^{sms,wfs}_{CC}(\mathcal{P}) \subseteq Cn^{sms,wfs}_{CB}(\mathcal{P})$$

Proof. (sketch) We want to show that: $Cn^{sms}_{CC}(\mathcal{P}) \subseteq Cn^{sms}_{CB}(\mathcal{P})$ and $Cn^{wfs}_{CC}(\mathcal{P}) \subseteq Cn^{wfs}_{CB}(\mathcal{P})$, respectively.

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 $floor$. Note that $Lit(floor)$ is a splitting set [15] for both $\mathcal{P}_{CB}$ and $\mathcal{P}_{CC}$ and thus the stable models for both $\mathcal{P}_{CB}$ and $\mathcal{P}_{CC}$ coincide on $floor$.

We can argue similarly for the well-founded semantics that the well-founded models of $\mathcal{P}_{CB}$ and $\mathcal{P}_{CC}$ coincide on $floor$, since both $\langle floor, \mathcal{P}_{CC} \setminus floor \rangle$ and $\langle floor, \mathcal{P}_{CB} \setminus floor \rangle$ are stratified pairs [18].

Moreover, $\mathcal{P}_{CB} \setminus floor$ and $\mathcal{P}_{CC} \setminus floor$ are (due to contextual boundedness) both positive logic programs modulo stratified input negation for literals from $floor$.[7] That means that both $\mathcal{P}_{CB} \setminus floor$ and $\mathcal{P}_{CC} \setminus floor$ extend the stable models (or the well-founded model[8]) of $floor$. Moreover, for each stable model (or the well-founded model) m of $floor$ and each rule $h@p : -l'_1, \ldots l'_n.$ which is satisfied in $\mathcal{P}_{CC} \setminus floor \cup m$ there is a corresponding rule $h@p : -l_1, \ldots l_n.$ which is satisfied in $\mathcal{P}_{CB} \setminus floor \cup m$. Finally, each of the remaining rules $h : -l_1, \ldots l_n.$ with an unscoped head in $\mathcal{P}_{CC} \setminus floor \cup m$ is also present in $\mathcal{P}_{CB} \setminus floor \cup m$. This proves that proving that the consequences of $\mathcal{P}_{CB}$ are a superset of the consequences of $\mathcal{P}_{CC}$ for both the well-founded and the stable semantics.

Indeed, there are consequences under contextually bounded semantics which are invalid under contextually closed semantics, for instance:

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

Here, the query $a?\{p\}$ is true with respect to contextually bounded semantics but not with respect to contextually closed semantics.

This reflects the intuition that contextually closed semantics draws inferences more cautiously and allow less interferences between programs, but does in trade not run into problems with contextually unbounded programs.

[7] By "input negation" we mean that given the stable models (or well-founded model) of $floor$ as "input fats" for $\mathcal{P}_{CB} \setminus floor$ and $\mathcal{P}_{CC} \setminus floor$ only these input facts can occur negatively in rule bodies.

[8] Since both well-founded semantics and stable model semantics coincide on stratified programs, we do not need to treat them separately for the remainder of the proof.

4 RDF(S) Plus Rules

In the Semantic Web, RDF is currently gaining more and more momentum. Thus, it is worthwhile to apply our context-aware rule language with scoped negation on arbitrary knowledge bases, consisting of RDF, RDFS and Logic Programming style rules with scoped negation distributed over the Web at different URIs. In this section we introduce a straightforward LP-compliant notion of a subset of RDF and investigate how it interacts with the semantics we have defined so far. To this end we will define a subset of the RDFS semantics in terms of open rules. As it turns out, we will need to slightly extend our definition of scoped literals to make it work. Based on this conclusion we will finally present a variant of contextually closed semantics in section 5.

We use a simplified syntax of RDF(S) here in terms of 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 [12] and partly shown in [8], large parts of RDF(S) can be embedded in logic programming. To this end, we use logic programs which express each statement $\langle S, P, O\rangle$ by a single atom `triple(S,P,O)`. Almost all of the RDFS semantics itself can be expressed by the following program

```
http://www.example.org/rdfs-semantics :
 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 axiomatic triples in RDF/RDFS, cf. [12, Sections 3.1 and 4.1]. For simplicity, we ignore XML literals, data types, containers and blank nodes here. Additional issues related with these features of RDF are out of the scope of this paper. We can now simply view the above program as a new context with the URI `http://www.example.org/rdfs-semantics` .

In order to illustrate the interplay of our semantics with this RDFS formulation, we revisit the examples from section 2 in terms of RDF, see Figure 2.

Our intention is to embed the RDFS semantics as a set of open rules in our framework. For this, we assume that an agent which answers a query is always aware of the RDFS context. However, as we will see, this is not enough.

http://www.moviereviews.com/
```
triple(ex:m1,ex:rate,ex:bad).
```

http://www.polleres.net/
```
triple(ex:m2,ex:rate,ex:bad).
triple(ex:m2,rdf:type,movie).
```

http://www.imdb.com/
```
triple(ex:m1,rdf:type,ex:sciFiMovie). triple(ex:m1,ex:title,"Plan 9 from Outer Space").
triple(ex:m1,ex:directedBy,"Ed Wood").
triple(ex:m2,rdf:type,ex:sciFiMovie). triple(ex:m2,ex:title,"Matrix Revolutions").
triple(ex:m2,ex:directedBy,"Andy Wachowski"). triple(ex:m2,ex:directedBy,"Larry
Wachowski").
triple(ex:m3,rdf:type,ex:sciFiMovie). triple(ex:m3,ex:title,"Bride of the Monster").
triple(ex:m3,ex:directedBy,"Ed Wood").
triple(ex:sciFiMovie,rdf:subClassOf,ex:movie).
...
```

Fig. 2. RDF versions of some of the programs from Figure 1

Note that negative literals that depend on RDFS inferences, immediately cause violations of contextual boundedness. Let us consider the query

"Give me all movies *not* listed at `http://www.imdb.com/`" (5)

asked to a search engine aware of contexts `http://www.example.org/rdfs-semantics`, `http://www.imdb.com/`, `http://www.moviereviews.com/`, and `http://www.polleres.net/`, cf. (2). The straightforward formulation of this query

```
answer(X) :- triple(X,rdf:type,ex:movie),
             not triple(X,rdf:type,ex:movie)@http://www.imdb.com/.
```

violates contextual boundedness because of the dependency between the negative literal from the query and the following RDFS rule:

```
triple(S,rdf:type,C2):- triple(S,rdf:type,C1),
                        triple(C1,rdfs:subClassOf,C2).
```

Now let us see how the same query is evaluated wrt. the contextually-closed semantics. We expect the answer to this query to be empty. However, the above-mentioned RDFS rule which should allow one to derive that `ex:m1` and `ex:m2` are movies listed by `http://www.imdb.com/`, will never be applied because of the working of tr_{CC}, and the final answer will be `ex:m2`.

We extend our syntax by allowing unions of contexts in literal scopes to deal with this problem, which allows us to reformulate the query above as follows:

"Give me all movies not listed at `http://www.imdb.com/`, *under additional consideration of* `http://www.example.org/rdfs-semantics`" (6)

which could be written as

```
answer(X) :- triple(X,rdf:type,ex:movie),
     not triple(X,rdf:type,ex:movie)@
     {http://www.example.org/rdfs-semantics, http://www.imdb.com/}.
```

We need to extend tr_{CC} to handle unions of contexts in scoped literals , as we will show in the next section. Note that we do not cover unions of contexts as an extension of tr_{CB} due to the inherent violation of contextual boundedness.

5 Contextually Closed Semantics with Context Sets

The basic intuition behind extending contextually closed semantics with unions of contexts in scoped literals is as follows: A literal scoped over a union of contexts shall be evaluated with respect to and closed over the union of the respective programs. Thus, we adapt the definition of tr_{CC} as follows:

Let $\mathcal{P}$ be an arbitrary set of programs, then $tr_{CC}(\mathcal{P})$ is defined by rewriting each rule $h : -l_1, \ldots, l_n.$ in any of the programs in $\mathcal{P}$ to $h@\mathcal{P} : -l'_1, \ldots, l'_n.$ where

$$l'_i = \begin{cases} l@\mathcal{R} & \text{in case } l_i = l@\mathcal{R} \text{ is a scoped literal with possibly set scope } \mathcal{R} \\ l_i@\mathcal{P} & \text{otherwise} \end{cases}$$

plus recursively adding $tr_{CC}(\mathcal{R})$ for any scoped body literal $l@\mathcal{R}$.

Note that this more general definition is equivalent to the original definition of tr_{CC} despite it per se includes the relevant part of the closure already. That means, we can also simplify the definition of $\mathcal{P}_{CC}$ in Definition 8 as follows:

$$\mathcal{P}_{CC} = \bigcup_{p \in (\mathcal{P})} (tr_{CC}(p) \cup p)$$

The remainder of Definition 8 can stay as is for this generalization. As we can easily verify, query (6) would be correctly answered under this semantics.

6 Related Works

FLORA-2 [13] is a query answering system based on the logic programming fragment of F-Logic [14], a frame-based syntactic variant of first-order logic popular for ontology reasoning. FLORA-2's module mechanism allows a form of scoped negation as failure using the well-founded semantics. Negative queries can be posed to a certain module. However, variables can be used in the place of the module identifier, in which case the scope of the query (negation) is the union of all the modules registered with the system at that point in time. This rather unrestricted way for defining scoped negation does not fulfill our monotonicity criterion with respect to the addition of new modules in the general case. 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. Implementations of our transformations on top of FLORA-2 are possible.

N3 [3] is a language for representing RDF rules on the Web. It has a form of scoped negation as failure and without an explicit notion of context. In N3 negation appears in the form of an infix operator `log:notIncludes`[9] that links two (possibly complex) formulas and that succeeds whenever the first formula does not include the second one. However, a formula is not necessarily closed in our sense and can have infinite size in N3 due to the presence of blank nodes

[9] `http://www.w3.org/2000/10/swap/doc/Reach`

(existentials) in the head of the rules.[10] This leads to a possibly infinite search space for negation as failure, which is undesirable and contradictive with the requirement of context-monotonicity.

TRIPLE [8] is another logic programming engine particularly tailored for RDF reasoning and querying with the support of scoping over possibly parameterized contexts, allowing union, intersection and set difference over contexts. The authors outline that nonmonotonic negation interpreted under the well-founded semantics can be supported. Since variables are allowed in parameterized contexts, similar considerations apply as for FLORA-2.

C-OWL [4] is a proposed extension of OWL by contexts and bridge rules. An interesting feature of this language is that its semantics makes use of so-called *local model semantics* where for each context there exists a local set of models and a local domain of interpretation. These kinds of semantics are opposed to the *global model semantics* where there exists a global model and a global domain of interpretation. Global model semantics have the disadvantage that local inconsistency propagates to the whole, which is not desirable on the Web. Our semantics follows a global model nature: When building on top of stable semantics local inconsistency propagates to the whole model. Note that the well-founded variants of our semantics do not involve inconsistency, and thus local inconsistencies cannot arise. Investigation of the relations and possible integrations with C-OWL are on our agenda.

Finally, we point out that the idea behind scoped negation as failure is orthogonal to the so-called local closed world assumption [9]. Instead of stating "local complete knowledge" as is done in local closed world assumption, we merely impose to explicitly close off any use of negation over a context, not making any statement about whether the knowledge of this context is indeed complete.

7 Conclusion

In this paper we discussed logic programs under contextually scoped negation and provided two possible semantics based on simple translations to normal logic programs. The rationale behind was keeping these translations lightweight in order to facilitate direct implementations with existing engines based on either the stable or well-founded semantics, while preserving context-monotonicity. Although our framework is general we emphasized the fruitful application of our approach in the context of rules on top of RDF and RDFS.

This work is a first step towards a proper definition for scoped negation for the upcoming RIF working group in W3C. Open issues like for instance the treatment of full RDF including blank nodes, data types, etc. require further investigation. Also, the transformations to normal logic programs provided in this paper still possibly contain redundant rules and might be subject to optimizations for actual implementations. As for future extensions, it seems to be useful to extend our definition of scope not only to unions but also intersections or

[10] `http://lists.w3.org/Archives/Public/public-cwm-bugs/2005Jul/0000.html`

set difference of contexts. More refined concepts of context such as e.g. so-called named RDF graphs [6] could also 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 RDF query languages such as for instance SPARQL [17] and N3 [3] in this direction.

Acknowledgments. The authors thank Jos de Bruijn, Rubén Lara, and Michael Kifer for fruitful discussions and the anonymous reviewers for their useful feedback. This work is partially supported by the EC projects DIP, KnowledgeWeb, Inframebs, SEKT, and ASG; by the FIT-IT projects RW^2 and TSC; by SFI grant SFI/02/CE1/I13; by the CICyT project TIC-2003-9001-C02.

References

1. J. Angele et al. Web rule language (WRL). W3C Member Submission, `http://www.w3.org/Submission/WRL/`, June 2005.
2. S. Battle, et al. Semantic Web services Framework (SWSF). W3C Member Submission, `http://www.w3.org/Submission/SWSF/`, 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, Apr. 2005.
4. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *Second International Semantic Web Conference - ISWC 2003*, Sanibel Island, FL, USA, 2003.
5. D. Brickley, R. V. Guha (eds.), and B. McBride (series ed.). RDF Vocabulary Description Language 1.0. Feb. 2004. W3C Recommendation, `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.
6. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs. *Journal of Web Semantics*, 3(4), 2005.
7. Paulo Pinheiro da Silva, Deborah L. McGuinness, and Rob McCool. Knowledge provenance infrastructure. *IEEE Data Engineering Bulletin*, 26(4), 2003.
8. S. Decker, M. Sintek, and W. Nejdl. The model-theoretic semantics of TRIPLE. Technical report, 2002.
9. O. Etzioni, K. Golden, and D. Weld. Tractable closed world reasoning with updates. In *KR'94: Principles of Knowledge Representation and Reasoning*, Bonn, Germany, 1994.
10. A. Van Gelder, K. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *7th ACM Symposium on Principles of Database Systems*, Austin, Texas, 1988.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *5th Int'l Conf. on Logic Programming*, Cambridge, Massachusetts, 1988.
12. P. Hayes. RDF semantics. W3C Recommendation, `http://www.w3.org/TR/rdf-mt/`, Feb. 2004.
13. M. Kifer. Nonmonotonic reasoning in FLORA-2. In *8th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, Diamante, Italy, 2005.

14. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4), 1995.
15. V. Lifschitz and H. Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *11th Int'l Conf. on Logic Programming (ICLP'94)*, Santa Margherita Ligure, Italy, June 1994.
16. D.L. McGuinness and F. van Harmelen. OWL Web ontology language overview. W3C Recommendation, `http://www.w3.org/TR/2004/REC-owl-features-20040210/`, Feb. 2004.
17. E. Prud'hommeaux and A. Seaborne (eds.). SPARQL Query Language for RDF, July 2005. W3C Working Draft.
18. J.S. Schlipf. Formalizing a Logic for Logic Programming. AMAI,5(2-4), 1992.