Rules with Contextually Scoped Negation for the Web

Axel Polleres

Universidad Rey Juan Carlos, Madrid
axel@polleres.net

March 6, 2006

Overview

- The Semantic Web
- Where to add Rules in the "Layer Cake"?
- A lightweight approach: Logic Programs with Context and Scoped Negation
  - Contextually Bounded Semantics
  - Contextually Closed Semantics
  - Summary/Open Issues
- Other approaches . . . time allowed.
  - SWRL – Rules on top of OWL
  - DLP – Intersection of LP and DL
  - dl-programs – a query interface between LP and OWL

Motivation - Semantic Web

http://imdb.com http://badmovies.org

- The Semantic Web promises machine readable **metadata** annotations of such sites allowing to combine and query their content, draw additional inferences.

Motivation - Semantic Web

http://imdb.com http://badmovies.org

- The Semantic Web promises machine readable **metadata** annotations of such sites allowing to combine and query their content, draw additional inferences.
- E.g., imagine a "Semantic" search engine gathering metadata on movies and ratings, using an agreed vocabulary, I want to ask **queries**, such as: *"Search for science fiction movies which are rated as bad?"*

Motivation - Semantic Web

http://imdb.com http://badmovies.org

- The Semantic Web promises machine readable **metadata** annotations of such sites allowing to combine and query their content, draw additional inferences.
- E.g., imagine a "Semantic" search engine gathering metadata on movies and ratings, using an agreed vocabulary, I want to ask **queries**, such as: *"Search for science fiction movies which are rated as bad?"*
- I want to express **taxonomies** such as *"Science-fiction movies are movies."*

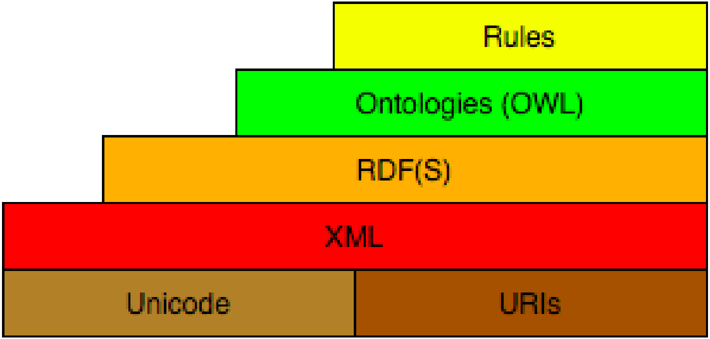
Motivation - Semantic Web

http://imdb.com

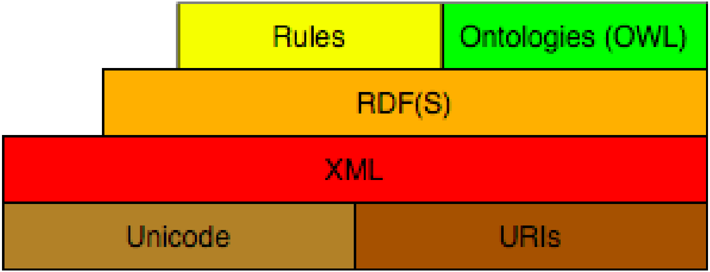
http://badmovies.org

- The Semantic Web promises machine readable **metadata** annotations of such sites allowing to combine and query their content, draw additional inferences.
- E.g., imagine a "Semantic" search engine gathering metadata on movies and ratings, using an agreed vocabulary, I want to ask **queries**, such as: *"Search for science fiction movies which are rated as bad?"*
- I want to express **taxonomies** such as *"Science-fiction movies are movies."*
- Besides **facts** in RDF, I want to express more complex **rules** such as for instance: *"All movies listed on* `badmovies.org` *are rated bad."*

The W3C's Semantic Web "layer cake"



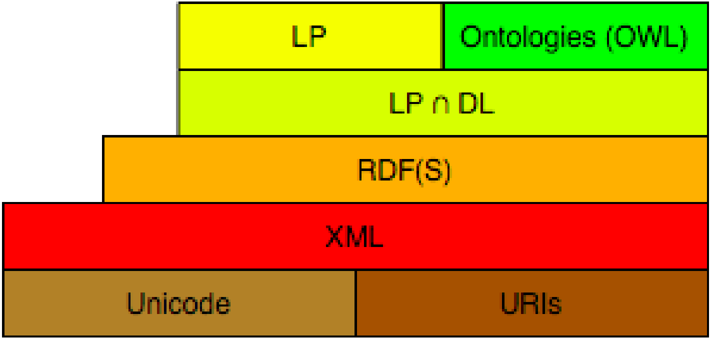
The W3C's Semantic Web "layer cake"



Can LP style rules really be layered ON TOP of OWL?

I. Horrocks , B. Parsia , P. Patel-Schneider , J. Hendler. *Semantic Web Architecture: Stack or Two Towers?* PPSWR, 2005.

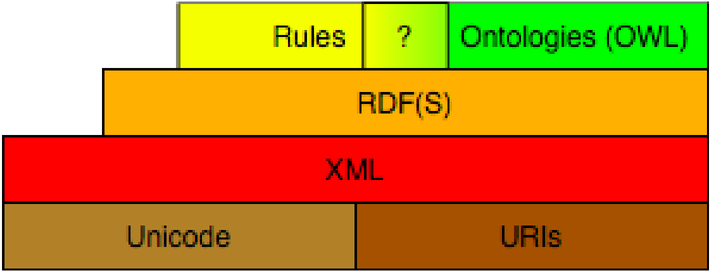
The W3C's Semantic Web "layer cake"



What is the common interoperability layer?

B. Grosof, I. Horrocks , R. Volz , S. Decker. *Description Logic Programs: Combining Logic Programs with Description Logic.* WWW, 2003.

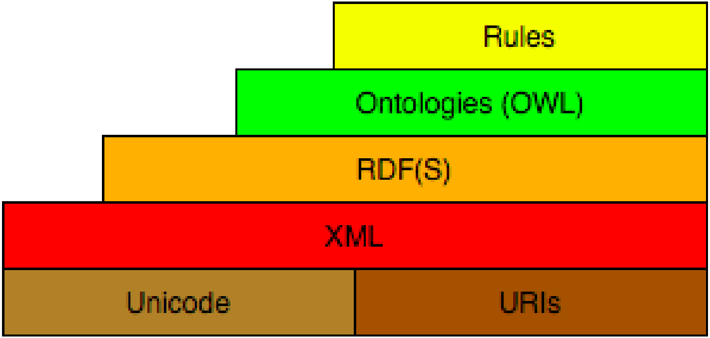
The W3C's Semantic Web "layer cake"



Can we define a "safe" interface between LP and OWL?

T. Eiter, T. Lukasiewicz, R. Schindlauer, H. Tompits *Combining Answer Set Programming with Description Logics for the Semantic Web.* KR, 2004.

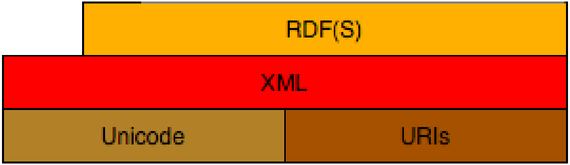
The W3C's Semantic Web "layer cake"



What is the "right" way to go?

RDF - A standard for metadata

Let's start at the level where concerns are still (more or less) clear:



- RDF allows to define *factual* metadata in about resources in form of triples

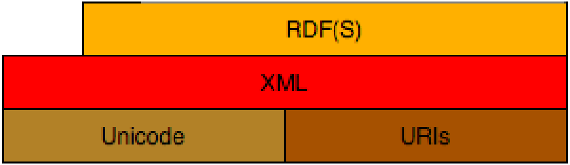
$$\langle Subject, Predicate, Object \rangle$$

  e.g. StarWars is directed by Goerge Lucas.

- Resources identified by URIs

RDF - A standard for metadata

Let's start at the level where concerns are still (more or less) clear:



- RDF allows to define *factual* metadata in about resources in form of triples

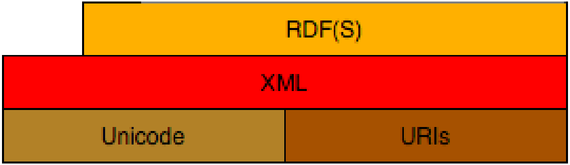
$$\langle Subject, Predicate, Object \rangle$$

  e.g. StarWars is directed by Goerge Lucas.

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`

RDF - A standard for metadata

Let's start at the level where concerns are still (more or less) clear:



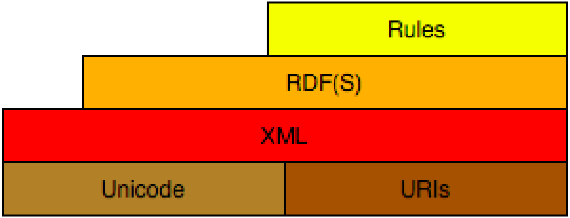
- RDF allows to define *factual* metadata in about resources in form of triples

$$\langle Subject, Predicate, Object \rangle$$

  e.g. StarWars is directed by Goerge Lucas.
- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type, rdf:subClassOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)

RDF - A standard for metadata

Let's start at the level where concerns are still (more or less) clear:



- RDF allows to define *factual* metadata in about resources in form of triples

$$\langle Subject, Predicate, Object \rangle$$

  e.g. StarWars is directed by Goerge Lucas.
- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type, rdf:subClassOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- The presented approach discuss rules on top of RDF(S) only.

Metadata on the Web as RDF facts.

http://moviereviews.com/
`triple(ex:m1,ex:rate,ex:bad).`
	http://polleres.net/myreviews
	`triple(ex:m2,ex:rate,ex:bad).`
	`triple(ex:m2,rdf:type,movie).`

http://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,rfd: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).
...
```

Figure: RDF triples for some movie information sites

RDFS semantics

RDFS semantics can (to a large extent) be captured by LP style rules:

```
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. Sections 3.1 and 4.1 of

http://www.w3.org/TR/rdf-mt/.

A lightweight approach to add rules:

Adding normal logic programs on top of RDF(S)

- We want to add arbitrary LP style rules on top of RDF(S)

A lightweight approach to add rules:

Adding normal logic programs on top of RDF(S)

- We want to add arbitrary LP style rules on top of RDF(S)
- We want to allow negation as failure (normal logic programs)

A lightweight approach to add rules:

Adding normal logic programs on top of RDF(S)

- We want to add arbitrary LP style rules on top of RDF(S)
- We want to allow negation as failure (normal logic programs)
- We want to base our semantics on the stable model semantics for logic programs

A lightweight approach to add rules:

Adding normal logic programs on top of RDF(S)

- We want to add arbitrary LP style rules on top of RDF(S)
- We want to allow negation as failure (normal logic programs)
- We want to base our semantics on the stable model semantics for logic programs
- But: There are some problems when allowing negation as failure on the Web

The stable model semantics for logic programs (1/2)

Syntax:
A normal logic programs P is a set of rules of the form:

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

- $l_1, \ldots, l_n$ are literals, i.e. atoms $p(t_1, \ldots, t_m)$ or negated atoms `not` $p(t_1, \ldots, t_m)$, such that $t_1, \ldots, t_m$ are either constants or variables.

The stable model semantics for logic programs (1/2)

Syntax:

A normal logic programs P is a set of rules of the form:

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

- $l_1, \ldots, l_n$ are literals, i.e. atoms $p(t_1, \ldots, t_m)$ or negated atoms `not` $p(t_1, \ldots, t_m)$, such that $t_1, \ldots, t_m$ are either constants or variables.
- h is an atom.

Semantics:

Herbrand models defined as usual:

- U_H consists of the the set of all constants appearing in P
- B_H is the set of all atoms constructible from predicate symbols in P and constants in U_H.
- Since there are no function symbols, B_H is finite.
- A Herbrand interpretation I is a subset of B_H.
- We denote by $ground(P)$ the set of all possible **ground instantiations** of rules in P where variables are substituted with the constants in U_H.
- A Herbrand interpretation I is called Herbrand model of P if all rules in $ground(P)$ are satisfied wrt. I.
- Each positive (`not`-free) program P has a unique minimal Herbrand model M.

The stable model semantics for logic programs (2/2)

The stable models for programs with negation is defined via the *Gelfond-Lifschitz-reduct*:
Let I be a Herbrand interpretation of P. Then the reduct P^I denotes the set of rules obtained from $ground(P)$ by

- removing all rules r such that `not` a occurs in the body of r for some $a \in I$
- rmoving all literals `not` a from the remaining rules.

The stable model semantics for logic programs (2/2)

The stable models for programs with negation is defined via the *Gelfond-Lifschitz-reduct*:
Let I be a Herbrand interpretation of P. Then the reduct P^I denotes the set of rules obtained from $ground(P)$ by

- removing all rules r such that `not` a occurs in the body of r for some $a \in I$
- rmoving all literals `not` a from the remaining rules.

A Herbrand interpretation M is called *stable model* of a normal logic program P iff M is the minimal Herbrand model of P^M.

The stable model semantics for logic programs (2/2)

The stable models for programs with negation is defined via the *Gelfond-Lifschitz-reduct*:
Let I be a Herbrand interpretation of P. Then the reduct P^I denotes the set of rules obtained from $ground(P)$ by

- removing all rules r such that `not` a occurs in the body of r for some $a \in I$
- rmoving all literals `not` a from the remaining rules.

A Herbrand interpretation M is called *stable model* of a normal logic program P iff M is the minimal Herbrand model of P^M.

There are efficient solvers to compute stable models: `dlv`, smodels, cmodels, assat, etc.

Incomplete knowledge on the Web

Problems:

- Incompleteness: The knowledge of a search engine about the Web is notoriously incomplete, i.e. it does not know about all available Websites.

  "Search for **all** *movies by Ed Wood"*

  Cannot be answered, without e.g. local completeness assumptions. Usually, this is not a problem as long as query results are good enough (sound, at least).

Incomplete knowledge on the Web

Problems:

- Incompleteness: The knowledge of a search engine about the Web is notoriously incomplete, i.e. it does not know about all available Websites.

  "Search for **all** *movies by Ed Wood"*

  Cannot be answered, without e.g. local completeness assumptions. Usually, this is not a problem as long as query results are good enough (sound, at least).

- More severe problems with negation in rules and queries:

  "Search for science fiction movies which are NOT rated as bad?"

  problematic, since using normal *negation as failure* over a finite subset of webpages is not only incomplete, but incorrect!

Incomplete knowledge on the Web

Problems:

- Incompleteness: The knowledge of a search engine about the Web is notoriously incomplete, i.e. it does not know about all available Websites.

  "Search for **all** *movies by Ed Wood"*

  Cannot be answered, without e.g. local completeness assumptions. Usually, this is not a problem as long as query results are good enough (sound, at least).

- More severe problems with negation in rules and queries:

  "Search for science fiction movies which are NOT rated as bad?"

  problematic, since using normal *negation as failure* over a finite subset of webpages is not only incomplete, but incorrect!

Solution: Enforce to make the scope for negation as failure always explicit!

Metadata on the Web as distributed rule sets

http://moviereviews.com/
`rated(m1,bad).`
`rated(X,bad) :-`
`    directedBy(X,"Ed Wood").`
	http://polleres.net/myreviews
	`rated(m2,bad).  movie(m2).`
	`rated(X,bad) :- movie(X),`
	`    not movie(X)@http://imdb.com.`
	
	http://badmovies.org/
	`movie(m1).`
	`...`
	`rated(X,bad) :- movie(X)@http://badmovies.org.`

http://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).
...
```

Figure: We use a more LP notation than before . . . and add rules

Syntax: Logic Programs with scoped literals

Assumption: A *program* is a set of rules associated with a URI u, where it is accessible:

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

- Body Literals:

Syntax: Logic Programs with scoped literals

Assumption: A *program* is a set of rules associated with a URI u, where it is accessible:

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

- Body Literals:
  - open (unscoped) literals a.

Syntax: Logic Programs with scoped literals

Assumption: A *program* is a set of rules associated with a URI u, where it is accessible:

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

- Body Literals:
  - open (unscoped) literals a.
  - scoped literals $(not)\ a@u$.

Syntax: Logic Programs with scoped literals

Assumption: A *program* is a set of rules associated with a URI u, where it is accessible:

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

- Body Literals:
  - open (unscoped) literals a.
  - scoped literals $(not)\ a@u$.
  - negative literals (negation as failure) MUST be scoped!

Syntax: Logic Programs with scoped literals

Assumption: A *program* is a set of rules associated with a URI u, where it is accessible:

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

- Body Literals:
  - open (unscoped) literals a.
  - scoped literals $(not)\ a@u$.
  - negative literals (negation as failure) MUST be scoped!
- Head atoms are always unscoped (to be viewed in the scope/context of the program (uri) where they appear), i.e. if h is derived from p we would also expect $h@p$ to be derivable.

Syntax: Logic Programs with scoped literals

Assumption: A *program* is a set of rules associated with a URI u, where it is accessible:

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

- Body Literals:
  - open (unscoped) literals a.
  - scoped literals $(not)\ a@u$.
  - negative literals (negation as failure) MUST be scoped!
- Head atoms are always unscoped (to be viewed in the scope/context of the program (uri) where they appear), i.e. if h is derived from p we would also expect $h@p$ to be derivable.

Examples of open and scoped rules:

```
http://moviereviews.com :
    rated(X,bad) :- directedBy(X,"Ed Wood").
http://badmovies.org :
    movie(m1).
    ...
    rated(X,bad) :- movie(X)@http://badmovies.org.
```

Requirements for a reasonable semantics for such rules

Let $Cn_{\mathcal{S}}(\mathcal{P})$ denote the set of conequences from a set of programs $\mathcal{P}$ wrt. semantics $\mathcal{S}$

Requirements for a reasonable semantics for such rules

Let $Cn_{\mathcal{S}}(\mathcal{P})$ denote the set of conequences from a set of programs $\mathcal{P}$ wrt. semantics $\mathcal{S}$

R1 **Context-Monotonicity**: When asking query q over (open and scoped) literals to an agent which is aware of a set of programs $\mathcal{P}$ (query context), I expect that I don't need to retract any inferences when asking another agent aware of $\mathcal{R} \supset \mathcal{P}$, i.e.

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

Requirements for a reasonable semantics for such rules

Let $Cn_{\mathcal{S}}(\mathcal{P})$ denote the set of conequences from a set of programs $\mathcal{P}$ wrt. semantics $\mathcal{S}$

R1 **Context-Monotonicity**: When asking query q over (open and scoped) literals to an agent which is aware of a set of programs $\mathcal{P}$ (query context), I expect that I don't need to retract any inferences when asking another agent aware of $\mathcal{R} \supset \mathcal{P}$, i.e.

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

R2 The chosen semantics should be **closed under context closure, i.e.**

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

where $Cl(\mathcal{P})$ is the set of all programs in $\mathcal{P}$ plus the ones "linked" recursively via scoped literals.

Requirements for a reasonable semantics for such rules

Let $Cn_{\mathcal{S}}(\mathcal{P})$ denote the set of conequences from a set of programs $\mathcal{P}$ wrt. semantics $\mathcal{S}$

R1 **Context-Monotonicity**: When asking query q over (open and scoped) literals to an agent which is aware of a set of programs $\mathcal{P}$ (query context), I expect that I don't need to retract any inferences when asking another agent aware of $\mathcal{R} \supset \mathcal{P}$, i.e.

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

R2 The chosen semantics should be **closed under context closure, i.e.**

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

where $Cl(\mathcal{P})$ is the set of all programs in $\mathcal{P}$ plus the ones "linked" recursively via scoped literals.

We define two semantics based on the stable model semantics, both fullfilling R1, one of them fullfilling R2.

Contextually Bounded semantics: Cn_{CB} (1/2)

Intuitively, scoping negative literals alone is not enough, since scoped literals can again depend on open rules, e.g.

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://moviereviews.com.
```

depends on whether the agent evaluating this rule knows `http://imdb.com` or not.

Contextually Bounded semantics: Cn_{CB} (1/2)

Intuitively, scoping negative literals alone is not enough, since scoped literals can again depend on open rules, e.g.

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://moviereviews.com.
```

depends on whether the agent evaluating this rule knows `http://imdb.com` or not.

1^{st} proposal to deal with this: Allow only contextually bounded negation.

> *We call a (set of) rules contextually bounded if no negative literal recursively depends on unscoped (open) literals.*

Contextually Bounded semantics: Cn_{CB} (1/2)

Intuitively, scoping negative literals alone is not enough, since scoped literals can again depend on open rules, e.g.

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://moviereviews.com.
```

depends on whether the agent evaluating this rule knows `http://imdb.com` or not.

1^{st} proposal to deal with this: Allow only contextually bounded negation.

> *We call a (set of) rules contextually bounded if no negative literal recursively depends on unscoped (open) literals.*

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://badmovies.com.
```

Contextually Bounded semantics: Cn_{CB} (1/2)

Intuitively, scoping negative literals alone is not enough, since scoped literals can again depend on open rules, e.g.

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://moviereviews.com.
```

depends on whether the agent evaluating this rule knows `http://imdb.com` or not.

1^{st} proposal to deal with this: Allow only contextually bounded negation.

> *We call a (set of) rules contextually bounded if no negative literal recursively depends on unscoped (open) literals.*

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://badmovies.com.
```

Attention: This rule is NOT contextually bounded:

```
rated(X,bad) :- movie(X), not movie(X)@http://imdb.com.
```

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://moviereviews.com.

interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://badmovies.com.
```

http://moviereviews.com/
`rated(m1,bad).`
`rated(X,bad) :-`
`    directedBy(X,"Ed Wood").`
	http://polleres.net/myreviews
	`rated(m2,bad).  movie(m2).`
	`rated(X,bad) :- movie(X),`
	`    not movie(X)@http://imdb.com.`
	
	http://badmovies.org/
	`movie(m1).`
	`...`
	`rated(X,bad) :- movie(X)@http://badmovies.org.`

```
interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://moviereviews.com.

interestingmovie(X) :- movie(X),
                       not rated(X,bad)@http://badmovies.com.
```

http://moviereviews.com/
`rated(m1,bad).`
`rated(X,bad) :-`
`    directedBy(X,"Ed Wood").`
	http://polleres.net/myreviews
	`rated(m2,bad).  movie(m2).`
	`rated(X,bad) :- movie(X),`
	`        not movie(X)@http://imdb.com.`
	
	http://badmovies.org/
	`movie(m1).`
	`...`
	`rated(X,bad) :- movie(X)@http://badmovies.org.`

http://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)@http://imdb.com.
...
```

Contextually Bounded semantics: Cn_{CB} (2/2)

Semantics is based on usual stable model semantics, plus "linking" open and closed literals by the following rewriting p_{CB} for each rule in program p:

$$h \texttt{ :- } l_1, \ \ldots, \ l_n.$$

Contextually Bounded semantics: Cn_{CB} (2/2)

Semantics is based on usual stable model semantics, plus "linking" open and closed literals by the following rewriting p_{CB} for each rule in program p:

$$h \texttt{ :- } l_1, \ \ldots, \ l_n.$$

$\Rightarrow$

$$h@p \texttt{ :- } l_1, \ \ldots, \ l_n.$$
$$h \texttt{ :- } h@p.$$

Contextually Bounded semantics: Cn_{CB} (2/2)

Semantics is based on usual stable model semantics, plus "linking" open and closed literals by the following rewriting p_{CB} for each rule in program p:

h `:-` l_1`,` ... `,` l_n`.`

$\Rightarrow$

$h@p$ `:-` l_1`,` ... `,` l_n`.`
h `:-` $h@p$`.`

Let $\mathcal{P}_{CB} = \bigcup_{p \in Cl(\mathcal{P})} p_{CB}$, then

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

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

Contextually Bounded semantics: Implications

R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).

Contextually Bounded semantics: Implications

R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).

R2 holds trivially ($\mathcal{P}_{CB}$ is defined via the closure of $\mathcal{P}$).

Contextually Bounded semantics: Implications

- R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).
- R2 holds trivially ($\mathcal{P}_{CB}$ is defined via the closure of $\mathcal{P}$).

Problem:

Contextually Bounded semantics: Implications

R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).

R2 holds trivially ($\mathcal{P}_{CB}$ is defined via the closure of $\mathcal{P}$).

Problem:

- Contextual boundedness is a prerequisite:

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

Contextually Bounded semantics: Implications

R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).

R2 holds trivially ($\mathcal{P}_{CB}$ is defined via the closure of $\mathcal{P}$).

Problem:

- Contextual boundedness is a prerequisite:

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

r:
```
c.
```

Contextually Bounded semantics: Implications

R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).

R2 holds trivially ($\mathcal{P}_{CB}$ is defined via the closure of $\mathcal{P}$).

Problem:

- Contextual boundedness is a prerequisite:

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

- Contextual boundedness is hardly maintainable in an open context, especially when contexts change (adding open rule):

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

Contextually Bounded semantics: Implications

R1 holds, by contextual boundedness (easy proof in [Polleres, et al. 2006]).

R2 holds trivially ($\mathcal{P}_{CB}$ is defined via the closure of $\mathcal{P}$).

Problem:

- Contextual boundedness is a prerequisite:

  p:

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

  r:

  ```
  c.
  ```

- Contextual boundedness is hardly maintainable in an open context, especially when contexts change (adding open rule):

  p:

  ```
  a :- c.
  ```

  r:

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

Contextually Closed semantics Cn_{CC} (1/2)

Alternative approach: Intuitively "close off", all open rules if referenced via a scoped literal.
We define an alternative rewriting p_{CC} for each rule in program p:

$$h \texttt{ :- } l_1, \ \ldots, \ l_n.$$

Contextually Closed semantics Cn_{CC} (1/2)

Alternative approach: Intuitively "close off", all open rules if referenced via a scoped literal.
We define an alternative rewriting p_{CC} for each rule in program p:

$$h \text{ :- } l_1, \ \ldots, \ l_n.$$

$\Rightarrow$

$$h@p \text{ :- } l'_1, \ \ldots, \ l'_n.$$

where $l'_i = l_i$ for scoped literals and $l'_i = l_i@p$ otherwise.

Contextually Closed semantics Cn_{CC} (1/2)

Alternative approach: Intuitively "close off", all open rules if referenced via a scoped literal.
We define an alternative rewriting p_{CC} for each rule in program p:

$$h \texttt{ :- } l_1, \ \ldots, \ l_n.$$

$\Rightarrow$

$$h@p \texttt{ :- } l'_1, \ \ldots, \ l'_n.$$

where $l'_i = l_i$ for scoped literals and $l'_i = l_i@p$ otherwise.
Let $\mathcal{P}_{CC} = \bigcup_{p \in \mathcal{P}} \cup \bigcup_{p \in Cl(\mathcal{P})} p_{CC}$, then

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

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

Contextually Closed semantics Cn_{CC} (2/2)

Intuitively, contextually closed semantics is more cautious or "local":

Contextually Closed semantics Cn_{CC} (2/2)

Intuitively, contextually closed semantics is more cautious or "local":

- $Cn_{CC}(\mathcal{P}) \subseteq Cn_{CB}(\mathcal{P})$ (proof in [Polleres, et al. 2006]).

Contextually Closed semantics Cn_{CC} (2/2)

Intuitively, contextually closed semantics is more cautious or "local":

- $Cn_{CC}(\mathcal{P}) \subseteq Cn_{CB}(\mathcal{P})$ (proof in [Polleres, et al. 2006]).
- It does not "traverse" closure concerning open literals, i.e. **R2** does not hold:

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

r:
```
c.
```

Here, $c \in Cn_{CB}(p)$, but $c \notin Cn_{CC}(p)$

Contextually Closed semantics Cn_{CC} (2/2)

Intuitively, contextually closed semantics is more cautious or "local":

- $Cn_{CC}(\mathcal{P}) \subseteq Cn_{CB}(\mathcal{P})$ (proof in [Polleres, et al. 2006]).
- It does not "traverse" closure concerning open literals, i.e. **R2** does not hold:

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

  c.
  ```
  Here, $a \in Cn_{CB}(p)$, but $a \notin Cn_{CC}(p)$ which one might consider more intuitive, i.e. cross-effects of open literals only within the query context.

Summary of this approach

- Can be used on top of RDF (modulo blank nodes)

Summary of this approach

- Can be used on top of RDF (modulo blank nodes)
- Clear definition of "scoped" negation

Summary of this approach

- Can be used on top of RDF (modulo blank nodes)
- Clear definition of "scoped" negation
- First attempt to
  - issue context-monotonicity
  - make dinstinction between open and closed rules

Summary of this approach

- Can be used on top of RDF (modulo blank nodes)
- Clear definition of “scoped” negation
- First attempt to
  - issue context-monotonicity
  - make dinstinction between open and closed rules
- The two solution proposed are simple/cautious on purpose, trying to start discussion about the “right” semantics of scoped negation for the Semantic Web.

Related works

- FLORA-2 (Kifer): an engine for F-Logic programs, allows modules, i.e. contexts, open literals/rules supported by allowing variables in place of modules, e.g.

  $$a : -b@X.$$

  No requirement for context-monotonicity though, well-founded semantics
- TRIPLE (Decker, et al.) allows parametrized contexts, union, intersection, set difference of contexts, also parameters allowed. Negation unsupported in current implementation, AFAIK.
- C-OWL extension of OWL by contexts and bridge rules, *local model semantics*, i.e. local inconsistencies do not spread over to the whole.

Sideremark: The approach is orthogonal to LCWA (Local closed world assumption) approaches allowing local completeness statements.

Issues/Future work

- Investigate a Local Model Semantics

Issues/Future work

- Investigate a Local Model Semantics
- Exact relation with SPARQL, RIF

Issues/Future work

- Investigate a Local Model Semantics
- Exact relation with SPARQL, RIF
- Complexity, Prototype implementation (DLV, YARS)

Issues/Future work

- Investigate a Local Model Semantics
- Exact relation with SPARQL, RIF
- Complexity, Prototype implementation (DLV, YARS)
- Investigate different semantics (well-founded vs. stable)

Issues/Future work

- Investigate a Local Model Semantics
- Exact relation with SPARQL, RIF
- Complexity, Prototype implementation (DLV, YARS)
- Investigate different semantics (well-founded vs. stable)
- Classical Negation, integration with the Ontology Layer (OWL)

Time allowed... How to integrate OWL with Rules?

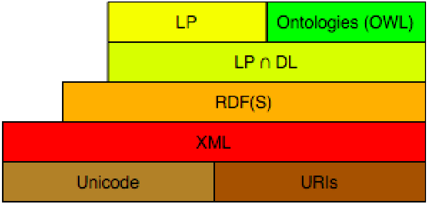
OWL (Web Ontology Language) adds more expressivity on top of RDF, allows to define taxonomies based on intersection, complement, cardinality restrictions, etc.

Axiom	DL Syntax
subClassOf	$C_1 \sqsubseteq C_2$
equivalentClass	$C_1 \equiv C_2$
disjointWith	$C_1 \sqsubseteq \neg C_2$
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$
subPropertyOf	$P_1 \sqsubseteq P_2$
equivalentProperty	$P_1 \equiv P_2$
inverseOf	$P_1 \equiv P_2^-$
transitiveProperty	$P^+ \sqsubseteq P$
functionalProperty	$\top \sqsubseteq \leqslant 1P$
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$

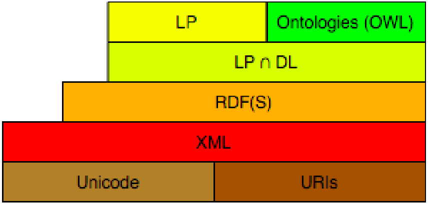
Constructor	DL Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$
complementOf	$\neg C$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$
allValuesFrom	$\forall P.C$
someValuesFrom	$\exists P.C$
maxCardinality	$\leqslant nP$
minCardinality	$\geqslant nP$

Expressivity in principle based on the description logic $\mathcal{SHOIN}(D)$.
(OWL DL, this is not not completely true for OWL Full)

Interoperability on the common (Horn) intersection only



Interoperability on the common (Horn) intersection only



DLP:

The Horn fragment of SHOIN(D) can be understood as a rule set. So, you can understand a small part of OWL as rules.

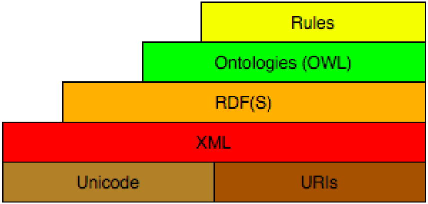
e.g. `father(X)` $\leftarrow$ `parent(X,Y),person(Y),male(X).`

$\Leftrightarrow \quad Father \sqsubseteq \exists Parent^{-1}.Human \sqcap Male$

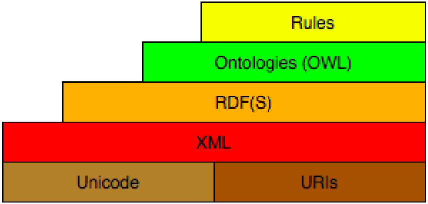
BUT: cannot cover much either on the rules part, nor on the DL part.

Only a basis for extensions in either direction.

Rules on top of OWL – SWRL



Rules on top of OWL – SWRL



SWRL:

Add Horn rules to OWL syntax, allows you to express e.g.

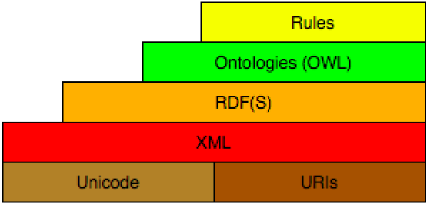
`uncle(X,Y)` $\leftarrow$ `male(X), sibling(X,Z),parent(Z,Y).`

But, also:

$\exists$`X parent(X,Y)` $\leftarrow$ `male(Z).`

(from $\exists Parent.Human \sqsubseteq male$)

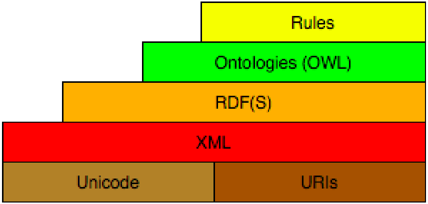
Rules on top of OWL – SWRL



SWRL:
Add Horn rules to OWL syntax, allows you to express e.g.
`uncle(X,Y)` $\leftarrow$ `male(X), sibling(X,Z),parent(Z,Y).`
But, also:
$\exists$`X parent(X,Y)` $\leftarrow$ `male(Z).`
(from $\exists Parent.Human \sqsubseteq male$)

- On the one hand naive combination of Horn + DL destroys decidability of either.

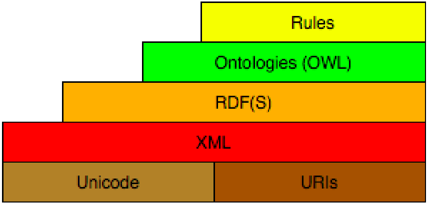
Rules on top of OWL – SWRL



SWRL:
Add Horn rules to OWL syntax, allows you to express e.g.
`uncle(X,Y)` $\leftarrow$ `male(X), sibling(X,Z),parent(Z,Y).`
But, also:
$\exists$`X parent(X,Y)` $\leftarrow$ `male(Z).`
(from $\exists Parent.Human \sqsubseteq male$)

- On the one hand naive combination of Horn + DL destroys decidability of either.
- On the other hand SWRL does not even allow arbitrary HORN but only binary/unary predicates.

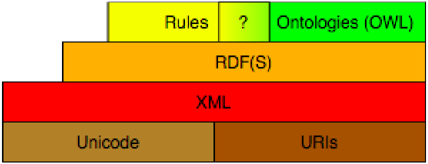
Rules on top of OWL – SWRL



SWRL:
Add Horn rules to OWL syntax, allows you to express e.g.
`uncle(X,Y)` $\leftarrow$ `male(X), sibling(X,Z),parent(Z,Y).`
But, also:
$\exists$`X parent(X,Y)` $\leftarrow$ `male(Z).`
(from $\exists Parent.Human \sqsubseteq male$)

- On the one hand naive combination of Horn + DL destroys decidability of either.
- On the other hand SWRL does not even allow arbitrary HORN but only binary/unary predicates.
- Issues like open vs. closed rules, negation as failure untouched.

Interface between LP and DL – dl-programs



T. Eiter, T. Lukasiewicz, R. Schindlauer, H. Tompits *Combining Answer Set Programming with Description Logics for the Semantic Web.* KR, 2004.

Define an extension of LP under the stable model semantics by so-called dl-atoms in the body, which allow to query a DL Knowledge base, but also interchange facts in the other direction. Authors define minimal Herbrand models and stable models for dl-programs.

- pro: Decidability remains.
- con: DL KB and LP program talk about different things, exchange only via "import/export".

Generalization of this technique available, HEX-programs.
Extension to scoped literals? Not straightforward.

Thank you for your attention!