Answer Set Programming for the Semantic Web

Tutorial

Thomas Eiter, Roman Schindlauer (TU Wien)
Giovambattista Ianni (TU Wien, Univ. della Calabria)
Axel Polleres (Univ. Rey Juan Carlos, Madrid)

Supported by IST REWERSE, FWF Project P17212-N04, CICyT project TIC-2003-9001-C02.

Unit 4 – Contribution of ASP to the Semantic Web

A. Polleres

Universidad Rey Juan Carlos, Madrid

European Semantic Web Conference 2006

Unit Outline

1. Introduction
2. ASP and RDF(S)
3. ASP and OWL
4. ASP and the Rules Layer

Goals of this Unit:

- Learn about overlaps and differences between ASP and SW Knowledge Representation Languages.
- Get introduced to related works in this area.
- Get an idea of how ASP can fruitfully extend these languages.

Unit Outline

Goals of this Unit:

- Learn about overlaps and differences between ASP and SW Knowledge Representation Languages.
- Get introduced to related works in this area.
- Get an idea of how ASP can fruitfully extend these languages.

Unit Outline

1. Introduction
2. ASP and RDF(S)
3. ASP and OWL
4. ASP and the Rules Layer

Goals of this Unit:

- Learn about overlaps and differences between ASP and SW Knowledge Representation Languages.
- Get introduced to related works in this area.
- Get an idea of how ASP can fruitfully extend these languages.

Unit Outline

1. Introduction
2. ASP and RDF(S)
3. ASP and OWL
4. ASP and the Rules Layer

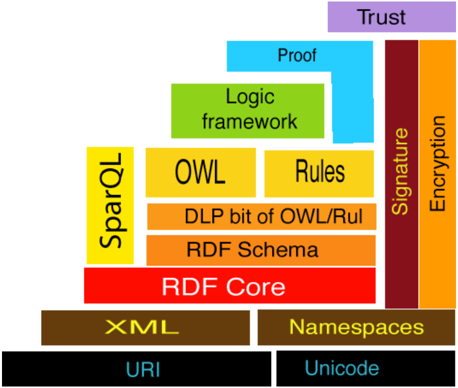
Goals of this Unit:

- Learn about overlaps and differences between ASP and SW Knowledge Representation Languages.
- Get introduced to related works in this area.
- Get an idea of how ASP can fruitfully extend these languages.

Introduction

In this unit, we give an overview of efforts and possibilities to deploy ASP related techniques in a Semantic Web context.

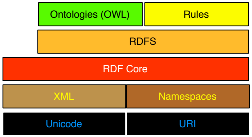
Question: Where does ASP fit in the “Layer Cake”?



Tim BL’s famous, layer cake, latest version [6]

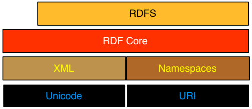
The three questions treated in this Unit:

1. ASP and RDF/RDFS:
    1. What of RDF/S can be expressed directly in ASP?
    2. What is different? Blank nodes, XML Literals, etc.
    3. RDF predicates in DLV (cf. Units 5 and 6)
2. ASP and OWL:
    1. What of OWL can be expressed directly in ASP?
    2. What is different? Existentials, number restrictions, equality reasoning, etc.
3. ASP and the Rules Layer
    1. General undecidability
    2. The "safe interaction" vs "safe interface"



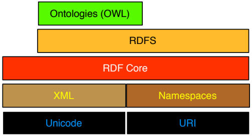
The three questions treated in this Unit:

1. ASP and RDF/RDFS:
   1. What of RDF/S can be expressed directly in ASP?
   2. What is different? Blank nodes, XML Literals, etc.
   3. RDF predicates in DLV (cf. Units 5 and 6)
2. ASP and OWL:
   1. What of OWL can be expressed directly in ASP?
   2. What is different? Existentials, number restrictions, equality reasoning, etc.
3. ASP and the Rules Layer
   1. General undecidability
   2. The “safe interaction” vs “safe interface”



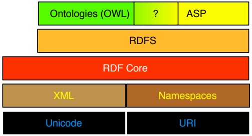
The three questions treated in this Unit:

1. ASP and RDF/RDFS:
    1. What of RDF/S can be expressed directly in ASP?
    2. What is different? Blank nodes, XML Literals, etc.
    3. RDF predicates in DLV (cf. Units 5 and 6)
2. ASP and OWL:
    1. What of OWL can be expressed directly in ASP?
    2. What is different? Existentials, number restrictions, equality reasoning, etc.
3. ASP and the Rules Layer
    1. General undecidability
    2. The "safe interaction" vs "safe interface"



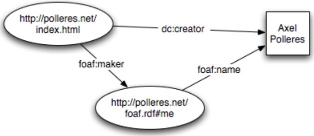
The three questions treated in this Unit:

1. ASP and RDF/RDFS:
   1. What of RDF/S can be expressed directly in ASP?
   2. What is different? Blank nodes, XML Literals, etc.
   3. RDF predicates in DLV (cf. Units 5 and 6)
2. ASP and OWL:
   1. What of OWL can be expressed directly in ASP?
   2. What is different? Existentials, number restrictions, equality reasoning, etc.
3. ASP and the Rules Layer
   1. General undecidability
   2. The “safe interaction” vs “safe interface”



What of RDF/S can be expressed directly in ASP?(1/2)

The RDF data model RDF describes a labeled graph of resources (nodes) linked to other resources or literals by predicates.



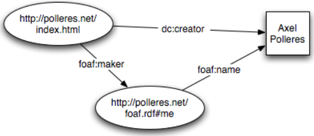
- usually represented in form of triples $\langle Subject, Predicate, Object \rangle$ e.g.

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me .
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

```
<rdf:Description rdf:about="http://polleres.net/index.html">
  <foaf:maker>
    <rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
      <foaf:Name>Axel Polleres</foaf:Name>
    </rdf:Description>
  </foaf:maker>
</rdf:Description>
```

What of RDF/S can be expressed directly in ASP?(1/2)

The RDF data model RDF describes a labeled graph of resources (nodes) linked to other resources or literals by predicates.



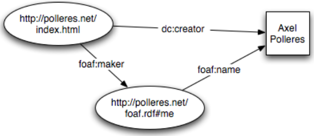
- usually represented in form of triples $\langle \textit{Subject}, \textit{Predicate}, \textit{Object} \rangle$ e.g.

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me .
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

```
<rdf:Description rdf:about="http://polleres.net/index.html">
  <foaf:maker>
    <rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
      <foaf:Name>Axel Polleres</foaf:Name>
    </rdf:Description>
  </foaf:maker>
</rdf:Description>
```

What of RDF/S can be expressed directly in ASP?(1/2)

The RDF data model RDF describes a labeled graph of resources (nodes) linked to other resources or literals by predicates.



- usually represented in form of triples $\langle Subject, Predicate, Object\rangle$ e.g.

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me.
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

```
<rdf:Description rdf:about="http://polleres.net/index.html">
  <foaf:maker>
    <rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
      <foaf:Name>Axel Polleres</foaf:Name>
    </rdf:Description>
  </foaf:maker>
</rdf:Description>
```

What of RDF/S can be expressed directly in ASP?(1/2)

RDF data model (cont'd):

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me.
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`,`rdfs:subPropertyOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- Common representation of RDF in ASP, use a ternary predicate:

```
triple("http://polleres.net/index.html","foaf:maker","http://polleres.net/foaf.rdf#me").
triple("http://polleres.net/foaf.rdf#me","foaf:name","Axel Polleres").
```

What of RDF/S can be expressed directly in ASP?(1/2)

RDF data model (cont'd):

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me.
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`,`rdfs:subPropertyOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- Common representation of RDF in ASP, use a ternary predicate:

```
triple("http://polleres.net/index.html","foaf:maker","http://polleres.net/foaf.rdf#me").
triple("http://polleres.net/foaf.rdf#me","foaf:name","Axel Polleres").
```

What of RDF/S can be expressed directly in ASP?(1/2)

RDF data model (cont'd):

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me.
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`,`rdfs:subPropertyOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- Common representation of RDF in ASP, use a ternary predicate:

```
triple("http://polleres.net/index.html","foaf:maker","http://polleres.net/foaf.rdf#me").
triple("http://polleres.net/foaf.rdf#me","foaf:name","Axel Polleres").
```

What of RDF/S can be expressed directly in ASP?(1/2)

RDF data model (cont'd):

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me.
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`,`rdfs:subPropertyOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- Common representation of RDF in ASP, use a ternary predicate:

```
triple("http://polleres.net/index.html","foaf:maker","http://polleres.net/foaf.rdf#me").
triple("http://polleres.net/foaf.rdf#me","foaf:name","Axel Polleres").
```

What of RDF/S can be expressed directly in ASP?(1/2)

RDF data model (cont'd):

```
http://polleres.net/index.html foaf:maker http://polleres.net/foaf.rdf#me .
http://polleres.net/foaf.rdf#me foaf:name "Axel Polleres"
```

- Resources identified by URIs
- RDFS allows to define simple taxonomies on RDF vocabularies using `rdf:type`, `rdf:subClassOf`,`rdfs:subPropertyOf`
- Some subtleties in RDF semantics (blank nodes, XML literals, RDF keywords treated as normal resources, reification, etc.)
- Common representation of RDF in ASP, use a ternary predicate:

```
triple("http://polleres.net/index.html","foaf:maker","http://polleres.net/foaf.rdf#me").
triple("http://polleres.net/foaf.rdf#me","foaf:name","Axel Polleres").
```

RDFS semantics can (to a large extent) be captured by ASP style 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 axiomatic triples in RDF/RDFS, cf. Sections 3.1 and 4.1 of http://www.w3.org/TR/rdf-mt/.

RDFS semantics can (to a large extent) be captured by ASP style 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 axiomatic triples in RDF/RDFS, cf. Sections 3.1 and 4.1 of http://www.w3.org/TR/rdf-mt/.

What is different in ASP compared with RDF/S?

- Blank nodes: Can usually be solved by newly generated Skolem-IDs (e.g. Raptor parser library uses this method.), also [74, 75] propose similar approach.

But: Be aware of UNA in ASP!

Example: GB and Axel both know Wolfgang: `knowing.rdf`

```
<rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
<rdf:Description rdf:about="http://www.gibbi.com/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
```

When we import these triples in an ASP and ask whether GB and Axel know different persons, we might come to false conclusions:

```
triple(X,Y,Z) :- &rdf["knowing.rdf"](X,Y,Z).
knowDifferentPeople(X,Y) :- triple(X,"foaf:knows",A),
                            triple(Y,"foaf:knows",B), A != B.
```

Will return

`(http://polleres.net/foaf.rdf#me, http://www.gibbi.com/foaf.rdf#me)`

as a valid pair.

Why? '`!=`' in ASP means "`not =`" (Negation as failure of proof!)

What is different in ASP compared with RDF/S?

- Blank nodes: Can usually be solved by newly generated Skolem-IDs (e.g. Raptor parser library uses this method.), also [74, 75] propose similar approach.

  But: Be aware of UNA in ASP!

  Example: GB and Axel both know Wolfgang: `knowing.rdf`

```
<rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
<rdf:Description rdf:about="http://www.gibbi.com/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
```

  When we import these triples in an ASP and ask whether GB and Axel know different persons, we might come to false conclusions:

```
triple(X,Y,Z) :- &rdf["knowing.rdf"](X,Y,Z).
knowDifferentPeople(X,Y) :- triple(X,"foaf:knows",A),
                            triple(Y,"foaf:knows",B), A != B.
```

  Will return

  `(http://polleres.net/foaf.rdf#me, http://www.gibbi.com/foaf.rdf#me)`

  as a valid pair.

  Why? '!=' in ASP means "not =" (Negation as failure of proof!)

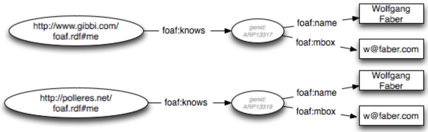
What is different in ASP compared with RDF/S?

- Blank nodes: Can usually be solved by newly generated Skolem-IDs (e.g. Raptor parser library uses this method.), also [74, 75] propose similar approach.

  But: Be aware of UNA in ASP!

  Example: GB and Axel both know Wolfgang: `knowing.rdf`

```
<rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
<rdf:Description rdf:about="http://www.gibbi.com/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
```



When we import these triples in an ASP and ask whether GB and Axel know different persons, we might come to false conclusions:

```
triple(X,Y,Z) :- &rdf["knowing.rdf"](X,Y,Z).
```

What is different in ASP compared with RDF/S?

- Blank nodes: Can usually be solved by newly generated Skolem-IDs (e.g. Raptor parser library uses this method.), also [74, 75] propose similar approach.

  But: Be aware of UNA in ASP!

  Example: GB and Axel both know Wolfgang: `knowing.rdf`

```
<rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
<rdf:Description rdf:about="http://www.gibbi.com/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
```

  When we import these triples in an ASP and ask whether GB and Axel know different persons, we might come to false conclusions:

```
triple(X,Y,Z) :- &rdf["knowing.rdf"](X,Y,Z).
knowDifferentPeople(X,Y) :- triple(X,"foaf:knows",A),
                            triple(Y,"foaf:knows",B), A != B.
```

  Will return

  `(http://polleres.net/foaf.rdf#me, http://www.gibbi.com/foaf.rdf#me)`

  as a valid pair.

  Why? '!=' in ASP means "not =" (Negation as failure of proof!)

What is different in ASP compared with RDF/S?

- Blank nodes: Can usually be solved by newly generated Skolem-IDs (e.g. Raptor parser library uses this method.), also [74, 75] propose similar approach.

  But: Be aware of UNA in ASP!

  Example: GB and Axel both know Wolfgang: `knowing.rdf`

```
<rdf:Description rdf:about="http://polleres.net/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
<rdf:Description rdf:about="http://www.gibbi.com/foaf.rdf#me">
  <foaf:knows><foaf:Person>
    <foaf:name>Wolfgang Faber</foaf:name>
    <foaf:mbox>w@faber.com</foaf:mbox>
  </foaf:Person></foaf:knows>
</rdf:Description>
```

  When we import these triples in an ASP and ask whether GB and Axel know different persons, we might come to false conclusions:

```
triple(X,Y,Z) :- &rdf["knowing.rdf"](X,Y,Z).
knowDifferentPeople(X,Y) :- triple(X,"foaf:knows",A),
                            triple(Y,"foaf:knows",B), A != B.
```

  Will return

  `(http://polleres.net/foaf.rdf#me, http://www.gibbi.com/foaf.rdf#me)`

  as a valid pair.

  Why? '`!=`' in ASP means "`not =`" (Negation as failure of proof!)

- RDFS has infinitely many axiomatic Triples, e.g.

```
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
```

  Strictly, speaking, that means that we would always need to deal with an *infinite* Herbrand Universe, when dealing with RDF.

- Note the difference: `rdfs:domain` and `rdfs:range` restrictions boiled down to RULES not to CONSTRAINTS. i.e.

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

  is NOT the same as:

```
:- triple(S,P,O), triple(P,rdfs:domain,C) not triple(S,rdf:type,C).
```

  However, often people rather intend to model constraints when using RDFS, see [10]

- RDFS has infinitely many axiomatic Triples, e.g.

```
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
```

  Strictly, speaking, that means that we would always need to deal with an *infinite* Herbrand Universe, when dealing with RDF.

- Note the difference: `rdfs:domain` and `rdfs:range` restrictions boiled down to RULES not to CONSTRAINTS. i.e.

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

  is NOT the same as:

```
:- triple(S,P,O), triple(P,rdfs:domain,C) not triple(S,rdf:type,C).
```

  However, often people rather intend to model constraints when using RDFS, see [10]

Training Example

Learn how to import RDF data into dlvhex:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from http://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

Training Example

Learn how to import RDF data into dlvhex:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from http://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

Training Example

Learn how to import RDF data into dlvhex:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from http://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

Training Example

Learn how to import RDF data into dlvhex:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from http://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

```
#namespace(foaf,"http://xmlns.com/foaf/0.1/")

knownByMe(X) :- &rdf["http://polleres.net/foaf.rdf"]
                ("http://polleres.net/foaf.rdf#me","foaf:knows",X).
```

Naive solution available as knowing2.dlh

Training Example

Learn how to import RDF data into dlvhex:

- Builtin for *namespace* definitions: #namespace(prefix,"URLinQuotes")
- Builtin for *RDF* import: &rdf[*URL*](X,Y,Z)

Task

Check the example knowing.dlh on the web page from the previous slide.

Try to modify knowing.dlh such that you extract from http://polleres.net/foaf.rdf the persons who "Axel Polleres" knows.

```
#namespace(foaf,"http://xmlns.com/foaf/0.1/")

triple(X,Y,Z) :- &rdf["http://polleres.net/foaf.rdf"](X,Y,Z).
knownbyMe(X) :- triple(ID,"foaf:name","Axel Polleres"),
                triple(ID,"foaf:knows",ID2),
                triple(ID2,"foaf:name",X).
```

A bit more elegant: Solution knowing3.dlh

ASP and OWL

- OWL offers more expressivity than RDF/S!
- What of OWL can be expressed directly in ASP?
- What is different? Existentials, number restrictions, equality reasoning, etc.
- Approaches for using ASP-style techniques for OWL reasoning Alsac and Baral [1], Swift [70], Hustadt,Motik,Sattler [45], Heymans et al. [42]

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(D)$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(D)$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(D)$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(D)$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/~axel$

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(\mathcal{D})$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(\mathcal{D})$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Facts

A large part of OWL (OWL DL) coincides with the Description Logics $\mathcal{SHOIN}(D)$.

Factual assertions (ABox):

1. Class membership (`rdf:type`) and property value assertions analogous to RDF. E.g.

```
<Paper rdf:ID="paper$_1$">
   <hasAuthor rdf:resource="thEiter">
</Paper>
```

$paper_1 \in Paper$,
$(paper_1, thEiter) \in hasAuthor$

2. Additional assertions in OWL: (In)equalities of individuals: `owl:sameAs`, `owl:differentFrom`, `owl:AllDifferent` E.g.

```
<rdf:Description about="http://polleres.net/">
   <owl:differentFrom rdf:resource="http://www.gibbi.com"/>
</rdf:Description>
<rdf:Description about="http://polleres.net/">
  <owl:sameAs rdf:resource="http://platon.escet.urjc.es/~axel"/>
</rdf:Description>
```

$polleres.net$
$\neq www.gibbi.com$
$polleres.net$
$= platon.escet.urjc.es/\ axel$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$$\geq 1 Year \sqsubseteq Paper$$
$$\top \sqsubseteq \forall Year.D_{xsd:integer}$$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$$isAuthorOf \equiv hasAuthor^{-}$$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$\geq 1 Year \sqsubseteq Paper$
$\top \sqsubseteq \forall Year.D_{xsd:integer}$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$isAuthorOf \equiv hasAuthor^{-}$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$\geq 1 Year \sqsubseteq Paper$

$\top \sqsubseteq \forall Year.D_{xsd:integer}$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$isAuthorOf \equiv hasAuthor^{-}$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$\geq 1 Year \sqsubseteq Paper$
$\top \sqsubseteq \forall Year.D_{xsd:integer}$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$isAuthorOf \equiv hasAuthor^{-}$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$$\geq 1 Year \sqsubseteq Paper$$
$$\top \sqsubseteq \forall Year.D_{xsd:integer}$$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$$isAuthorOf \equiv hasAuthor^{-}$$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$$\geq 1 Year \sqsubseteq Paper$$
$$\top \sqsubseteq \forall Year.D_{xsd:integer}$$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$$isAuthorOf \equiv hasAuthor^{-}$$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$$\geq 1 Year \sqsubseteq Paper$$
$$\top \sqsubseteq \forall Year.D_{xsd:integer}$$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$$isAuthorOf \equiv hasAuthor^{-}$$

OWL offers more expressivity than RDF/S - Properties

Structural axioms about Roles (RBox):

1. Datatype properties (having datatyes as range), e.g.

The property year has papers as its domain and xsd:integer as its range

```
<owl:DatatypeProperty rdf:ID="year">
   <rdfs:domain rdf:resource="#paper"/>
   <rdfs:range  rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

$$\geq 1 Year \sqsubseteq Paper$$
$$\top \sqsubseteq \forall Year.D_{xsd:integer}$$

2. Object properties (having classes as range) – analogously.
3. Defining inverse, transitive, or symmetric properties, e.g.

"isAuthorOf" is the inverse of "hasAuthor"

```
<owl:ObjectProperty rdf:ID="isAuthorOf">
   <owl:inverseOf rdf:resource="#hasAuthor"/>
</owl:ObjectProperty>
```

$$isAuthorOf \equiv hasAuthor^{-}$$

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

A senior researcher is a person who is author of more than 3 papers some of which valid publications

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$$

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

A senior researcher is a person who is author of more than 3 papers some of which valid publications

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$$

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

A senior researcher is a person who is author of more than 3 papers some of which valid publications

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$$

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

*A senior researcher is a **person** who is author of more than 3 papers some of which valid publications*

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$$

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

A senior researcher is a person who is author of more than 3 papers some of which valid publications

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$$

OWL offers more expressivity than RDF/S - Complex Class definitions

Structural axioms about Classes (TBox): Complex Class definitions in OWL beyond `rdfs:subclassOf`:

1. by **union** of other classes, e.g. $Reviewers \sqcup Senior \sqsubseteq PCMember$
2. by **intersection** of other classes: e.g. $Professor \sqsubseteq Researcher \sqcap Teacher$
3. by **property restrictions**: e.g. $\exists isAuthorOf.JournalArticle \sqcup \leq 5 isPCMemberOf \sqsubseteq Senior$
4. by **enumerations** of individuals: e.g. $Color \sqsubseteq \{red, green, blue\}$

A more complex example:

A senior researcher is a person who is author of more than 3 papers some of which valid publications

```
<owl:Class rdf:ID="senior">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#person"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:minCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
      3
      </owl:minCardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isAuthorOf"/>
      <owl:someValuesFrom rdf:resource="#publication"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

$$Senior \equiv Person \sqcap \geq 3 isAuthorOf \sqcap \exists isAuthorOf.Publication$$

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1 R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ... A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1 R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1 R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	$\texttt{Paper(paper}_1\texttt{).}$
$(paper_1, thEiter) \in hasAuthor$	$\texttt{hasAuthor(paper}_1\texttt{,thEiter).}$

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^-.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^-$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^-$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	$\texttt{A(X) :- C}_1\texttt{(X),..., C}_n\texttt{(X).}$
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$\texttt{C}_1\texttt{(X) :- A(X). ...C}_n\texttt{(X) :- A(X).}$
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	$\texttt{C}_1\texttt{(X) v ...v C}_n\texttt{(X) :- A(X).}$
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	$\texttt{A(X) :- C}_1\texttt{(X). ... A(X) :- C}_n\texttt{(X).}$

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^-.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^-$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^-$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^-.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^-$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^-$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^-.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^-$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^-$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1 R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ... A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ... A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^-.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^-$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^-$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1 R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts "as is":

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^* \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1 R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL can be expressed directly in ASP?

We restrict ourselves to OWL DL here, and also use DL syntax for its easier legibility.

ABox factual knowledge about Class membership and property values and can be translated to ASP facts “as is”:

DL syntax	Intuitive correspondence with ASP rules/facts
$paper_1 \in Paper$	`Paper(paper`$_1$`).`
$(paper_1, thEiter) \in hasAuthor$	`hasAuthor(paper`$_1$`,thEiter).`

RBox/TBox: A subset of OWL can be straightforwardly translated to ASP:

DL syntax	Intuitive correspondence with ASP rules/facts
$\top \sqsubseteq \forall R^{-}.A$ (rdfs:domain)	`A(X) :- R(X,Y).`
$\top \sqsubseteq \forall R.A$ (rdfs:range)	`A(Y) :- R(X,Y).`
$\top \sqsubseteq \forall R.Datatype$ (rdfs:range)	`:- R(X,Y), not &datatype(Y).` where `&datatype` is a builtin datatype predicate
$R \sqsubseteq S$ (rdfs:subPropertyOf)	`S(X,Y) :- R(X,Y).`
$R^{*} \sqsubseteq R$ (owl:transitiveProperty)	`R(X,Z) :- R(X,Y), R(Y,Z).`
$R \equiv R^{-}$ (owl:symmetricProperty)	`R(X,Y) :- R(Y,X).`
$R \equiv S^{-}$ (owl:inversOf)	`R(X,Y) :- S(Y,X).`
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$	`A(X) :- C`$_1$`(X),..., C`$_n$`(X).`
$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	`C`$_1$`(X) :- A(X). ...C`$_n$`(X) :- A(X).`
$\exists R.C \sqsubseteq A$ (owl:someValuesFrom, lhs)	`A(X) :- R(X,Y),C(Y).`
$\geq 1R \sqsubseteq A$ (owl:minCardinality 1, lhs)	`A(X) :- R(X,Y).`
$A \sqsubseteq \forall R.C$ (owl:allValuesFrom, rhs)	`C(Y) :- R(X,Y),A(X).`
$A \sqsubseteq C_1 \sqcup \ldots \sqcup C_n$ (owl:unionOf rhs)	`C`$_1$`(X) v ...v C`$_n$`(X) :- A(X).`
$C_1 \sqcup \ldots \sqcup C_n \sqsubseteq A$ (owl:unionOf lhs)	`A(X) :- C`$_1$`(X). ...A(X) :- C`$_n$`(X).`

What of OWL cannot be expressed directly in ASP?

$A \equiv \{o_1, \ldots, o_n\}$ (owl:oneOf)	Cannot be directly translated... only approximated non-modularly if equality predicates allowed in rule bodies.
$A \sqsubseteq \bot$ (owl:Nothing)	`:- A(X).` is an Approximation only, doesn't work for complex concepts!

$A \sqsubseteq \exists R.C$ (owl:someValuesFrom rhs)	Impossible, we have no existentials in rule heads
$\forall R.C \subseteq A$ (owl:allValuesFrom lhs)	One might guess: `A(X) :- not noRC(X). noRC(X) :- R(X,Y), -C(Y).` but doesn't work :-(
cardinality restrictions, owl:sameAs, owl:differentFrom	Need reasoning with equality, expensive to implement. Recall: "=" and "!=" are not classical equality but builtin syntactic equality (UNA,CWA)!
...etc.	

What of OWL cannot be expressed directly in ASP?

$A \equiv \{o_1, \ldots, o_n\}$ (owl:oneOf)	Cannot be directly translated... only approximated non-modularly if equality predicates allowed in rule bodies.
$A \sqsubseteq \bot$ (owl:Nothing)	`:- A(X).` is an Approximation only, doesn't work for complex concepts!

$A \sqsubseteq \exists R.C$ (owl:someValuesFrom rhs)	Impossible, we have no existentials in rule heads
$\forall R.C \subseteq A$ (owl:allValuesFrom lhs)	One might guess: `A(X) :- not noRC(X). noRC(X) :- R(X,Y), -C(Y).` but doesn't work :-(
cardinality restrictions, owl:sameAs, owl:differentFrom	Need reasoning with equality, expensive to implement. Recall: "=" and "!=" are not classical equality but builtin syntactic equality (UNA,CWA)!

...etc.

What of OWL cannot be expressed directly in ASP?

$A \equiv \{o_1, \ldots, o_n\}$ (owl:oneOf)	Cannot be directly translated... only approximated non-modularly if equality predicates allowed in rule bodies.
$A \sqsubseteq \bot$ (owl:Nothing)	`:- A(X).` is an Approximation only, doesn't work for complex concepts!

$A \sqsubseteq \exists R.C$ (owl:someValuesFrom rhs)	Impossible, we have no existentials in rule heads
$\forall R.C \subseteq A$ (owl:allValuesFrom lhs)	One might guess: `A(X) :- not noRC(X). noRC(X) :- R(X,Y), -C(Y).` but doesn't work :-(
cardinality restrictions, owl:sameAs, owl:differentFrom	Need reasoning with equality, expensive to implement. Recall: "=" and "!=" are not classical equality but builtin syntactic equality (UNA,CWA)!
...etc.	

Main differences OWL vs. ASP?

- `not` in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - `not`: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$\neg Publication \sqsubseteq Unpublished$	`Unpublished(X) :- not Publication(X).`
$paper_1 \in Paper.$	`Paper(paper`$_1$`).`
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: `Unpublished(paper`$_1$`).`

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$axel \in \neg Paper.$	`-Paper(axel).`
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: `-Publication(axel).`

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, **disjunction or unstratified negation are necessary!**

But: not enough! ASPs is strong query answering, algorithms not tailored for e.g. subsumption checking like DL's.

Main differences OWL vs. ASP?

- not in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - not: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$\neg Publication \sqsubseteq Unpublished$	`Unpublished(X) :- not Publication(X).`
$paper_1 \in Paper.$	Paper(paper$_1$).
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: Unpublished(paper$_1$).

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$axel \in \neg Paper.$	`-Paper(axel).`
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: `-Publication(axel).`

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, **disjunction or unstratified negation are necessary!**

But: not enough! ASPs is strong query answering, algorithms not tailored for e.g. subsumption checking like DL's.

Main differences OWL vs. ASP?

- not in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - not: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$\neg Publication \sqsubseteq Unpublished$	`Unpublished(X) :- not Publication(X).`
$paper_1 \in Paper.$	`Paper(`$paper_1$`).`
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: `Unpublished(`$paper_1$`).`

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$axel \in \neg Paper.$	`-Paper(axel).`
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: `-Publication(axel).`

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, **disjunction or unstratified negation are necessary!**

But: not enough! ASPs is strong query answering, algorithms not tailored for e.g. subsumption checking like DL's.

Main differences OWL vs. ASP?

- `not` in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - `not`: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$\neg Publication \sqsubseteq Unpublished$	`Unpublished(X) :- not Publication(X).`
$paper_1 \in Paper.$	`Paper(paper`$_1$`).`
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: `Unpublished(paper`$_1$`).`

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$axel \in \neg Paper.$	`-Paper(axel).`
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: `-Publication(axel).`

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, disjunction or unstratified negation are necessary!

But: not enough! ASPs is strong query answering, algorithms not tailored for e.g. subsumption checking like DL's.

Main differences OWL vs. ASP?

- not in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - not: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$\neg Publication \sqsubseteq Unpublished$	`Unpublished(X) :- not Publication(X).`
$paper_1 \in Paper.$	Paper($\mathtt{paper_1}$).
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: Unpublished($\mathtt{paper_1}$).

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	`Paper(X) :- Publication(X).`
$axel \in \neg Paper.$	`-Paper(axel).`
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: `-Publication(axel).`

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, **disjunction or unstratified negation** are necessary!

But: not enough! ASPs is strong query answering, algorithms not tailored for e.g. subsumption checking like DL's.

Main differences OWL vs. ASP?

- not in ASP is different from negation (owl:complementOf) in OWL:
  - $\neg$: Classical negation! Open world assumption! Monotonicity!
  - not: Different purpose! Closed world assumption! Non-monotonicity!

$Publication \sqsubseteq Paper$	Paper(X) :- Publication(X).
$\neg Publication \sqsubseteq Unpublished$	Unpublished(X) :- not Publication(X).
$paper_1 \in Paper.$	Paper(paper$_1$).
in DL: $\not\models paper_1 \in Unpublished$	Does infer in ASP: Unpublished(paper$_1$).

- Also strong negation in ASP is not completely the same as classical negation in DLs, e.g.

$Publication \sqsubseteq Paper$	Paper(X) :- Publication(X).
$axel \in \neg Paper.$	-Paper(axel).
in DL: $\models axel \in \neg Publication$	Does not infer in ASP: -Publication(axel).

Why? "Tertium non datur" does not hold in ASP!
What would we need to add? D(x) v -D(x).
$\Rightarrow$: In order to emulate DL, **disjunction or unstratified negation** are necessary!

But: not enough! ASPs is strong query answering, algorithms not tailored for e.g. subsumption checking like DL's.

ASP for OWL reasoning? (1/4)

Several approaches in the literature [1, 70, 60, 45, 41, 42, 12], some of which we will discuss here in brief.

1) Alsac and Baral[1]: Encodes the Description Logics $\mathcal{ALCQI}$ in ASP.

- Realizes that naive translation is insufficient.
- Embedding in nondisjunctive ASP, using guesses by unstratified negation to emulate classical behavior, e.g.
  `paper(X ) :- top(X ), not -paper(X).`
  `-paper(X ) :- top(X ), not paper(X ).`
- Facts are encoded as constraints, e.g.
  `:- not Paper(paper`$_1$`)`, instead of simply `Paper(paper`$_1$`)`..
- Similarly Inclusion axioms $\sqsubseteq$ encoded as constraints, e.g.
  $Publication \sqsubseteq Paper$ becomes `:- Publication(X), not Paper(X).`
- for complex class descriptions, new predicates symbols are introduced, e.g.

Problems:

- Keeps UNA (but so do prominent DL Reasoners like Racer (can be switched off), and FACT)
- Tailored for entailing facts assertions, function symbols needed for the general case, to emulate infinite domain (not supported by current ASP implementations).
- Not extensible to nominals in restrictions and enumerated classes. to emulate infinite domain (not supported by current ASP implementations).

ASP for OWL reasoning? (1/4)

Several approaches in the literature [1, 70, 60, 45, 41, 42, 12], some of which we will discuss here in brief.

1) Alsac and Baral[1]: Encodes the Description Logics $\mathcal{ALCQI}$ in ASP.

- Realizes that naive translation is insufficient.
- Embedding in nondisjunctive ASP, using guesses by unstratified negation to emulate classical behavior, e.g.
  `paper(X ) :- top(X ), not -paper(X).`
  `-paper(X ) :- top(X ), not paper(X ).`
- Facts are encoded as constraints, e.g.
  `:- not Paper(paper`$_1$`)`, instead of simply `Paper(paper`$_1$`).`.
- Similarly Inclusion axioms $\sqsubseteq$ encoded as constraints, e.g.
  $Publication \sqsubseteq Paper$ becomes `:- Publication(X), not Paper(X).`
- for complex class descriptions, new predicates symbols are introduced, e.g.

Problems:

- Keeps UNA (but so do prominent DL Reasoners like Racer (can be switched off), and FACT)
- Tailored for entailing facts assertions, function symbols needed for the general case, to emulate infinite domain (not supported by current ASP implementations).
- Not extensible to nominals in restrictions and enumerated classes. to emulate infinite domain (not supported by current ASP implementations).

ASP for OWL reasoning? (2/4)

2) Heymans, et al.[41] use a similar encoding of the DL $\mathcal{ALCHOQ}$, but with disjunction and "Open" answer sets.

- also keeps UNA
- no function symbols needed for the general case, instead relies on the (in general undecidable) open answer set semantics, which allows infinite, "open" domains.
- Evaluation algorithms and reductions of to existing ASP engines for decidable subsets described in [40].
- support for nominals and enumerated class again limited,

ASP for OWL reasoning? (3/4)

3) KAON approach to reduce DL reasoning to disjunctive Logic Programming, originally introduced by Motik et al. [60, 45], underlies the KAON2 system.

Remarks:

- Original approach was based on a limited translation of DL into disjunctive rules, including function symbols and a new predicate symbol for any complex class expression.
- Further optimized and developed [45] in the KAON2 system:
  - Novel implementation, not based on existing ASP solvers.
  - intermediate translation to first-order logic, clausal form transformation, function symbol elimination,
  - Algorithm based on basic superposition calculus for equality reasoning, to overcome UNA.
  - Disjunctive Logic Programming as "encoding" of DL with the goal of an alternative OWL DL reasoner.
  - not really ASP in the sense presented in this Tutorial, to some extent at the cost of declarativity.
  - Also probably not extensible to nominals.
  - cf. Tutorial on KAON2 @ this conference!

ASP for OWL reasoning? (4/4)

Summary:

- OWL does not really "fit" into ASP as such.
- Lossless encoding all of OWL into ASP is not only difficult, but also looses much of the declarativity and legibility of both formalisms (DL and ASP) for Knowledge Representation.

⇒ **Better:**
Aim at combining OWL and ASP for more powerful KR for the Web!

- Still, an active research area from which interesting extensions of ASP itself (Open ASPs, Superposition Calculus for equality reasoning etc.) arise!

ASP for OWL reasoning? (4/4)

Summary:

- OWL does not really "fit" into ASP as such.
- Lossless encoding all of OWL into ASP is not only difficult, but also looses much of the declarativity and legibility of both formalisms (DL and ASP) for Knowledge Representation.

$\Rightarrow$ **Better:**
Aim at combining OWL and ASP for more powerful KR for the Web!

- Still, an active research area from which interesting extensions of ASP itself (Open ASPs, Superposition Calculus for equality reasoning etc.) arise!

ASP for OWL reasoning? (4/4)

Summary:

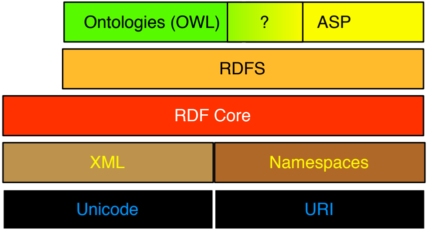
- OWL does not really "fit" into ASP as such.
- Lossless encoding all of OWL into ASP is not only difficult, but also looses much of the declarativity and legibility of both formalisms (DL and ASP) for Knowledge Representation.

$\Rightarrow$ **Better:**
Aim at combining OWL and ASP for more powerful KR for the Web!

- Still, an active research area from which interesting extensions of ASP itself (Open ASPs, Superposition Calculus for equality reasoning etc.) arise!

ASP and the rules Layer

⇒ **Better:**
Aim at combining OWL and ASP for more powerful KR for the Web!
ASP itself might be a good candidate for building a foundation of the rules layer!



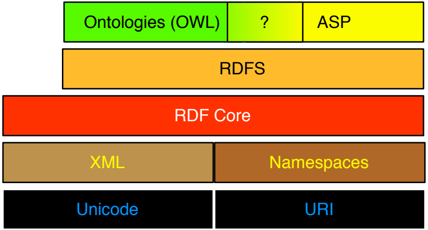
But: It's not THAT easy!

ASP and the rules Layer

⇒ **Better:**

Aim at combining OWL and ASP for more powerful KR for the Web!

ASP itself might be a good candidate for building a foundation of the rules layer!



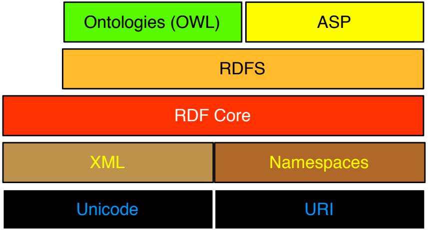
But: It's not THAT easy!

ASP and the rules Layer

⇒ **Better:**

Aim at combining OWL and ASP for more powerful KR for the Web!

ASP itself might be a good candidate for building a foundation of the rules layer!



But: It's not THAT easy!

ASP and the rules Layer

- Obstacles in Integrating ASP and Ontologies: Logic Programming vs. Classical Logic
  - Non-monotonicity of rules (Open world vs Closed World).
  - Equality vs. UNA.
  - Non-ground entailment.
- Strategies for combining rules and ontologies
  - Simple approaches
  - Safe interaction
  - Safe interface

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure *not* with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure not with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure *not* with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure *not* with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure not with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

Non-monotonicity of rules (Open world vs Closed World), equality

- As we've seen, it is not straightforward how to integrate constraints and negation as failure not with classical negation.
- Thus, we need a way to cater for both: Classical negation in the Ontology part and naf in the rules part.
- Moreover, we have seen discrepancies between UNA deployed in logic programming and equality in DLs.
- At least, for positive, non-disjunctive rules, without equality statements, everything seems clear... these have a pendant in classical logic: (function-free) Horn Clauses!

BUT...

... BUT: Non-ground entailment.

A set of Horn clauses is not the same as the corresponding logic program:

- Recall: Logic Programming based semantics of ASP is defined in terms of minimal Herbrand models, i.e., sets of ground facts.

$$\forall X \;\; \text{potableLiquid}(X) \leftarrow \text{wine}(X)$$
$$\forall X \;\; \text{wine}(X) \leftarrow \text{whiteWine}(X)$$
$$\text{whiteWine}(\text{"Welschriesling"})$$

- Both the LP reading and the Horn clause reading of this yield the entailment of facts *whiteWine("WelschRiesling"), wine("WelschRiesling"), potableLiquid("WelschRiesling")*.
- The Horn clauses furhter entail:

$$\text{wine}(\text{"WelschRiesling"}) \leftarrow \text{potableLiquid}(\text{"WelschRiesling"}),$$
$$\forall\ X\ .\text{whiteWine}(X) \leftarrow \text{PotableLiquid}(X).$$

- Logic Programs do not entail rules or other axioms, but only facts!

… BUT: Non-ground entailment.

A set of Horn clauses is not the same as the corresponding logic program:

- Recall: Logic Programming based semantics of ASP is defined in terms of minimal Herbrand models, i.e., sets of ground facts.

$$\begin{array}{l} \forall X \;\; \text{potableLiquid}(X) \leftarrow \text{wine}(X) \\ \forall X \;\; \text{wine}(X) \leftarrow \text{whiteWine}(X) \\ \text{whiteWine}(\text{"Welschriesling"}) \end{array}$$

- Both the LP reading and the Horn clause reading of this yield the entailment of facts
whiteWine("WelschRiesling"), wine("WelschRiesling"), potableLiquid("WelschRiesling").

- The Horn clauses furhter entail:

$$\begin{array}{l} \text{wine}(\text{"WelschRiesling"}) \leftarrow \text{potableLiquid}(\text{"WelschRiesling"}), \\ \forall\ X\ .\text{whiteWine}(X) \leftarrow \text{PotableLiquid}(X). \end{array}$$

- Logic Programs do not entail rules or other axioms, but only facts!

... BUT: Non-ground entailment.

A set of Horn clauses is not the same as the corresponding logic program:

- Recall: Logic Programming based semantics of ASP is defined in terms of minimal Herbrand models, i.e., sets of ground facts.

$$\forall X \quad \text{potableLiquid}(X) \leftarrow \text{wine}(X)$$
$$\forall X \quad \text{wine}(X) \leftarrow \text{whiteWine}(X)$$
$$\text{whiteWine}(\text{"Welschriesling"})$$

- Both the LP reading and the Horn clause reading of this yield the entailment of facts *whiteWine("WelschRiesling"), wine("WelschRiesling"), potableLiquid("WelschRiesling").*
- The Horn clauses furhter entail:

$$\text{wine}(\text{"WelschRiesling"}) \leftarrow \text{potableLiquid}(\text{"WelschRiesling"}),$$
$$\forall\ X\ .\text{whiteWine}(X) \leftarrow \text{PotableLiquid}(X).$$

- Logic Programs do not entail rules or other axioms, but only facts!

... BUT: Non-ground entailment.

A set of Horn clauses is not the same as the corresponding logic program:

- Recall: Logic Programming based semantics of ASP is defined in terms of minimal Herbrand models, i.e., sets of ground facts.

$$\forall X \ \ \mathrm{potableLiquid}(X) \leftarrow \mathrm{wine}(X)$$
$$\forall X \ \ \mathrm{wine}(X) \leftarrow \mathrm{whiteWine}(X)$$
$$\mathrm{whiteWine}(\text{"Welschriesling"})$$

- Both the LP reading and the Horn clause reading of this yield the entailment of facts *whiteWine("WelschRiesling"), wine("WelschRiesling"), potableLiquid("WelschRiesling").*
- The Horn clauses furhter entail:

$$\mathrm{wine}(\text{"WelschRiesling"}) \leftarrow \mathrm{potableLiquid}(\text{"WelschRiesling"}),$$
$$\forall \ X \ .\mathrm{whiteWine}(X) \leftarrow \mathrm{PotableLiquid}(X).$$

- Logic Programs do not entail rules or other axioms, but only facts!

… Non-ground entailment doesn't work for LP? So let's take Horn Logic for the Rules Layer!

SWRL [44] takes this approach:

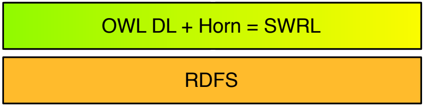
- extends OWL DL with
- Horn rules using unary and binary atoms representing classes (concepts) and roles (properties):

  shareFood(W1,W2) $\leftarrow$ hasDrink(D,W1), hasDrink(D,W2)

  Whitewine $\sqsubseteq$ Wine

  "Trout grilled" $\in$ Dish

  ("Trout grilled","WelschRiesling") $\in$ hasDrink



- But: Entailemnt for such a naive combination of Horn and DL is undecidable![52] :-(

… Non-ground entailment doesn't work for LP? So let's take Horn Logic for the Rules Layer!

SWRL [44] takes this approach:

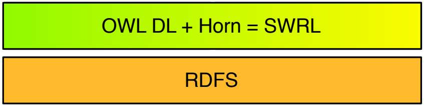
- extends OWL DL with
- Horn rules using unary and binary atoms representing classes (concepts) and roles (properties):

  shareFood(W1,W2) $\leftarrow$ hasDrink(D,W1), hasDrink(D,W2)

  Whitewine $\sqsubseteq$ Wine

  "Trout grilled" $\in$ Dish

  ("Trout grilled","WelschRiesling") $\in$ hasDrink



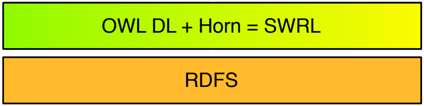
- But: Entailemnt for such a naive combination of Horn and DL is undecidable![52] :-(

... Non-ground entailment doesn't work for LP? So let's take Horn Logic for the Rules Layer!

SWRL [44] takes this approach:

- extends OWL DL with
- Horn rules using unary and binary atoms representing classes (concepts) and roles (properties):

$$\mathrm{shareFood}(W1,W2) \leftarrow \mathrm{hasDrink}(D,W1),\ \mathrm{hasDrink}(D,W2)$$
$$\mathrm{Whitewine} \sqsubseteq \mathrm{Wine}$$
$$\text{"Trout grilled"} \in \mathrm{Dish}$$
$$(\text{"Trout grilled"},\text{"WelschRiesling"}) \in \mathrm{hasDrink}$$



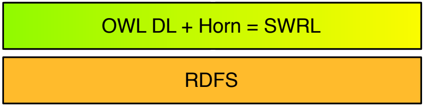
- But: Entailemnt for such a naive combination of Horn and DL is undecidable![52] :-(

... Non-ground entailment doesn't work for LP? So let's take Horn Logic for the Rules Layer!

SWRL [44] takes this approach:

- extends OWL DL with
- Horn rules using unary and binary atoms representing classes (concepts) and roles (properties):

$$\text{shareFood(W1,W2)} \leftarrow \text{hasDrink(D,W1), hasDrink(D,W2)}$$
$$\text{Whitewine} \sqsubseteq \text{Wine}$$
$$\text{"Trout grilled"} \in \text{Dish}$$
$$\text{("Trout grilled","WelschRiesling")} \in \text{hasDrink}$$



- But: Entailemnt for such a naive combination of Horn and DL is undecidable![52] :-(

... Non-ground entailment doesn't work for LP? So let's take Horn Logic for the Rules Layer!

SWRL [44] takes this approach:

- extends OWL DL with
- Horn rules using unary and binary atoms representing classes (concepts) and roles (properties):

$$\mathrm{shareFood(W1,W2)} \leftarrow \mathrm{hasDrink(D,W1),\ hasDrink(D,W2)}$$
$$\mathrm{Whitewine} \sqsubseteq \mathrm{Wine}$$
$$\text{"Trout grilled"} \in \mathrm{Dish}$$
$$(\text{"Trout grilled"},\text{"WelschRiesling"}) \in \mathrm{hasDrink}$$

OWL DL + Horn = SWRL

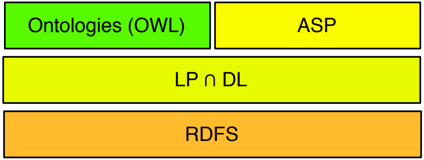
RDFS

- But: Entailemnt for such a naive combination of Horn and DL is undecidable![52] :-(

... Non-ground entailment, so what?

⇒ At least, we can say: Ontology reading and LP reading can interact on exchange of ground facts in the Horn **intersection** of OWL and ASP:

- i.e. the Horn fragment of $\mathcal{SHOIN}(D)$.



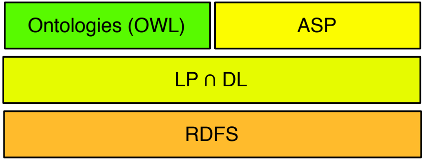
See DLP [39], and WRL [2] which extends DLP towards some features of ASP [1]

[1] not precisely true since WRL uses well-founded semantics

... Non-ground entailment, so what?

⇒ At least, we can say: Ontology reading and LP reading can interact on exchange of ground facts in the Horn **intersection** of OWL and ASP:

- i.e. the Horn fragment of $\mathcal{SHOIN}(D)$.



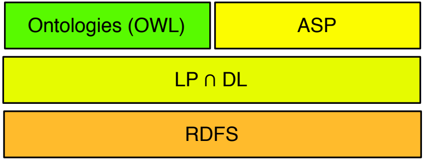
See DLP [39], and WRL [2] which extends DLP towards some features of ASP [1]

[1] not precisely true since WRL uses well-founded semantics

… Non-ground entailment, so what?

$\Rightarrow$ At least, we can say: Ontology reading and LP reading can interact on exchange of ground facts in the Horn **intersection** of OWL and ASP:

- i.e. the Horn fragment of $\mathcal{SHOIN}(D)$.



See DLP [39], and WRL [2] which extends DLP towards some features of ASP [1]

[1] not precisely true since WRL uses well-founded semantics

Well, there must be something in between, no?

Two basic approaches to retain decidability beyond DLP:

"Safe interaction" Rules interact with Ontologies in a common semantic framework, with syntactic restrictions

"Safe interface" Rules and Ontologies are kept strictly separate and only communicate via a *"safe interface"*, but do not impose syntactic restrictions on either the rules or the ontology part

Safe Interaction between LP and DL 1/2

Unrestricted recursive rules on top of DL cause the trouble of SWRL.

$\mathcal{AL}$-Log [20], extends the DL $\mathcal{AL}$ by Horn rules, with additional "safety" restriction:

> Every variable of a rule must appear in at least one of the rule atoms occurring in the body of R, where rule atoms are those predicates which do not appear in the DL Knowledge Base part, but only in rules.

This retains decidability!

Rules:

shareFood(W1,W2) $\leftarrow$ hasDrink(D,W1), hasDrink(D,W2), myWines(W1),myWines(W1).

cellarWine("Welchriesling"). cellarWine("Veltliner"). cellarWine("Zweigelt").

Ontology:

Whitewine $\sqsubseteq$ Wine
"Trout grilled" $\in$ Dish

$\vdots$

Safe Interaction between LP and DL 1/2

Unrestricted recursive rules on top of DL cause the trouble of SWRL.

$\mathcal{AL}$-Log [20], extends the DL $\mathcal{AL}$ by Horn rules, with additional "safety" restriction:

> Every variable of a rule must appear in at least one of the rule atoms occurring in the body of R, where rule atoms are those predicates which do not appear in the DL Knowledge Base part, but only in rules.

This retains decidability!

Rules:

shareFood(W1,W2) ← hasDrink(D,W1), hasDrink(D,W2),
myWines(W1),myWines(W1).

cellarWine("Welchriesling"). cellarWine("Veltliner"). cellarWine("Zweigelt").

Ontology:

Whitewine $\sqsubseteq$ Wine
"Trout grilled" $\in$ Dish

⋮

Safe Interaction between LP and DL 1/2

Unrestricted recursive rules on top of DL cause the trouble of SWRL.

$\mathcal{AL}$-Log [20], extends the DL $\mathcal{AL}$ by Horn rules, with additional "safety" restriction:

> Every variable of a rule must appear in at least one of the rule atoms occurring in the body of R, where rule atoms are those predicates which do not appear in the DL Knowledge Base part, but only in rules.

This retains decidability!

Rules:

shareFood(W1,W2) ← hasDrink(D,W1), hasDrink(D,W2),
myWines(W1),myWines(W1).
cellarWine("Welchriesling"). cellarWine("Veltliner"). cellarWine("Zweigelt").

Ontology:

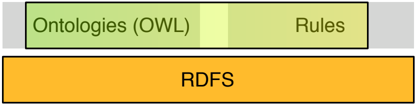
Whitewine $\sqsubseteq$ Wine
"Trout grilled" $\in$ Dish

$\vdots$

Safe Interaction between LP and DL 2/2

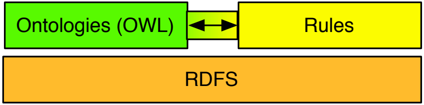
- The decidability for such so-called DL-safe rules was extended to $\mathcal{SHIQ}$ in Motik et al. [59]
- Heymans et al. [42] show decidability for query answering in $\mathcal{ALCHOQ}(\sqcup, \sqcap)$ DL-safe rules
- Rosati [65, 66] loosens the safety restriction further, by
  - allowing non-rule atoms also in rule heads, and
  - also provides an ASP style semantics for non-Horn rules.

All these approaches restrict either the DL, or rules or both:



Safe Interface between LP and DL – dl-programs

- *dl-programs* [27] Define an extension of ASP by so-called dl-atoms in rule bodies body, which allow to query a DL Reasoner, but also interchange facts in the other direction:



  - Decidability remains.
  - Full OWL DL and full ASP with all its extensions.
- Another approach in this direction: TRIPLE's [19] ability to query DL engines.

More on this in unit 5 and 6.

Questiontime...

Let's proceed with Unit 5!

Questiontime. . .

Let's proceed with Unit 5!