Inference Rules

olivier.corby@inria.fr
Inference Rules

IF A THEN B

A => B
Inference Rules (1)

grandParent(?x, ?z) :-

     parent(?x, ?y) & parent(?y ?z)

parent(John, Mary)

parent(Mary, James)

=> grandParent(John, James)
Inference Rules (2)

\[ ?y \ ?q \ ?x : - \]

\[ ?p \ \text{owl:inverseOf} \ ?q \ & \]

\[ ?x \ ?p \ ?y \]
Inference Rules (3)

\(?x\) rdf:type ex:Adult : -

\(?x\) a ex:Person &

\(?x\) ex:age ?age &

?age >= 18
SPARQL Construct-Where Rules

PAT1 :- PAT2

construct { PAT1 }

where { PAT2 }
Construct-where Inference Rules

construct { ?x a ex:Adult }

where {

    ?x a ex:Person .


    filter(?age >= 18)

}
Construct-where Inference Rules

1. Compute SPARQL query on *where* clause

2. For each query result:
   - instantiate *construct* template with result bindings
   - insert triples into graph
Construct-where Inference Rules

construct {

  ?x a ex:Adult

}

where {

  ?x a ex:Person .


  filter(?age >= 18)

}
Construct-where Inference Rules

```plaintext
construct { 
  ?x a ex:Adult 
}

where { 
  ?x a ex:Person . 
  filter(?age >= 18) 
}
```

Diagram:
- John is of type ex:Person.
- John has an age of 20.
Construct-where Inference Rules

construct {
    ?x a ex:Adult
}
where {
    ?x a ex:Person .
    filter(?age >= 18)
}
Construct-where Inference Rules

```
construct {
    ?x a ex:Adult
}
where {
    ?x a ex:Person .
    filter(?age >= 18)
}
```
Construct-where Inference Rules

construct {
  ?x a ex:Adult
}

where {
  ?x a ex:Person .
  filter(?age >= 18)
}
Construct-where Inference Rules

construct {
    ?x a ex:Adult
}
where {
    ?x a ex:Person .
    filter(?age >= 18)
}
Construct-where Inference Rules

Entailments are processed until **saturation**

- Loop on all Rules
- Until nothing new is deduced
Construct-where Inference Rules

Inferences stored in named graph $kg:\text{rule}$

Retrieve inferences:

```sql
select *
from $kg:\text{rule}$
where {
  ?x ?p ?y
}
```
Example: Symmetry

construct {
  ?y ?p ?x
}

where {
  ?p a owl:SymmetricProperty
  ?x ?p ?y
}
Example: Transitivity

```
construct {
    ?x ?p ?z
}

where {
    ?p rdf:type owl:TransitiveProperty
    ?x ?p ?y
    ?y ?p ?z
}
```
Exercise

Rules for rdfs:domain, rdfs:subPropertyOf
Example: domain

construct {
    ?x rdf:type ?d
}

where {
    ?p rdfs:domain ?d
    ?x ?p ?y
}
Example: subPropertyOf

construct {

  ?x ?q ?y
}

where {

  ?p rdfs:subPropertyOf ?q

  ?x ?p ?y
}
RDF/XML Rule Syntax

<rule>
<body>

prefix h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
construct {
    ?f h:hasSpouse ?x
}
where {
    ?x h:hasSpouse ?f
}

</body>
</rule>
RDF/XML Rule Syntax

<rule>
<body>

prefix h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
construct {
    ?f h:hasSpouse ?x
}
where {
    ?x h:hasSpouse ?f
}

</body>
</rule>
RDF/XML Rule Syntax

```xml
<rule>
  <body>
<!-- [CDATA[
prefix h: <http://www.inria.fr/2007/09/11/humans.rdfs#>
construct { 
  ?f h:hasSpouse ?x
}
where { 
  ?x h:hasSpouse ?f
}
]]>
</body>
</rule>
```
RDF/XML Rule Syntax

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [ 
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#"> 
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#"> 
<!ENTITY rul "http://ns.inria.fr/edelweiss/2011/rule#"> ]>

<rdf:RDF xmlns:rdfs="&rdfs;" xmlns:rdf="&rdf;" xmlns = '&rul;' >

<rule>
  <body>
    construct {} where {}
  </body>
</rule>

</rdf:RDF>