On RDF Validation, Stardog ICV, and Assorted Remarks

Evren Sirin, Kendall Clark

{evren,kendall}@clarkparsia.com

CLARK Parsia
Stardog RDF Database

- **State-of-the-art RDF database**
  - A fast, commercial, transactional, pure Java RDF database (quad store)

- **Feature rich**
  - Client-server & embeddable
  - Jena, Sesame, SNARL, HTTP interfaces
  - ACID transactions
  - Command-line and web admin interfaces
  - Role-based access control
  - Query-time reasoning with OWL and SWRL rules
  - Integrity Constraint Validation
Integrity Constraints (IC)

● Constraints define what is valid data
  ○ Can be stored in the database
  ○ Can be external to the database

● Two different modes for validation
  ○ On-demand mode
    ■ At any time, validate data stored in the database
    ■ Allow invalid data but be informed
  ○ Guard mode
    ■ Check constraints at commit time (insert/delete)
    ■ Commit fails if a constraint is violated
    ■ Database always guaranteed to have valid data
Stardog Integrity Constraints (IC)

- RDF Instance Data - for assertions
- OWL/SWRL ontology - for reasoning
- IC constraints - for validation
Stardog Integrity Constraints (IC)

- RDF Instance Data - for assertions
- OWL/SWRL ontology - for reasoning
- IC constraints - for validation
  - High-level, RDF-based concise syntax
  - Constraints are translated to SPARQL for execution
  - You can also write constraints in SPARQL directly
Stardog Integrity Constraints (IC)

- RDF Instance Data - for assertions
- OWL/SWRL ontology - for reasoning
- IC constraints - for validation
  - High-level, RDF-based concise syntax
  - Constraints are translated to SPARQL for execution
  - You can also write constraints in SPARQL directly
  - High-level syntax happens to be OWL/SWRL
SKOS Example

# SKOS reference ontology that defines inference rules
skos:broaderTransitive rdf:type owl:TransitiveProperty
skos:broader rdfs:subPropertyOf skos:broaderTransitive

# Constraints from SKOS reference expressed as ICs
skos:related owl:propertyDisjointWith skos:broaderTransitive

# SKOS data that violates the SKOS data model
:B skos:broader :C .
Why we used OWL syntax

- High-level syntax is essential
  - Concision and abstraction
  - Usability, understandability, maintenance
- Many validation constructs already in OWL
  - Domain/range, cardinality, uniqueness, disjointness, conjunctions, disjunctions, negation, ...
- Many people already think RDFS and OWL can be used for validation
  - But semantics not suitable for validation
  - So we defined constraint semantics for OWL axioms
Simple Constraint Example

Natural language: Every supervisor should supervise at least one employee

OWL Constraint: `Supervisor` subclassOf `supervises` some `Employee`

SPARQL Constraint:
```
SELECT *
WHERE {
  ?x type Supervisor .
  FILTER NOT EXISTS {
    ?x supervises ?y .
    ?y type Employee .
  }
}
```
Terp syntax for Constraints

Natural language

Every supervisor should supervise at least one employee

Terp syntax - Turtle syntax extended with OWL shortcuts using Manchester syntax

OWL Constraint

Supervisor subClassOf supervises some Employee

SPARQL Constraint

SELECT *
{
    ?x type Supervisor .
    FILTER NOT EXISTS {
        ?x supervises ?y .
        ?y type Employee .
    }
}
Simple Constraint Example

NATURAL LANGUAGE
Every supervisor should supervise at least one employee

OWL CONSTRAINT
Supervisor subClassOf supervises some Employee

SPARQL CONSTRAINT
```
SELECT *
{
  ?x type Supervisor .
  FILTER NOT EXISTS {
    ?x supervises ?y .
    ?y type Employee .
  }
}
```
Rule Syntax for Constraints

Natural language: Every supervisor should supervise at least one employee

OWL Constraint: Supervisor subClassOf supervises some Employee

SPARQL Constraint:
```
IF {
    ?x type Supervisor .
}
THEN {
    ?x supervises ?y .
    ?y type Employee .
}
```
## Complex Example

<table>
<thead>
<tr>
<th>Natural language</th>
<th>OWL Constraint</th>
<th>SPARQL Constraint</th>
</tr>
</thead>
</table>
| If a project is funded by only internal funding sources then it should be approved by the internal budget office. | `Project and (fundedBy only InternalFundingSource) subClassOf approvedBy value InternalBudgetOffice` | ```
SELECT * WHERE {
  ?x type Project .
  FILTER NOT EXISTS {
    ?x fundedBy ?y .
    FILTER NOT EXISTS {
      ?y type InternalFundingSource .
    }
  }
  }
  FILTER NOT EXISTS {
    ?x approvedBy InternalBudgetOffice .
  }
} ``` |
Reasoning and Validation

- RDF validation can be over...
  - an explicit RDF graph, or
  - an RDF graph under the semantics of a SPARQL 1.1 entailment regime

- Reasoning might cause a violation
  - See SKOS example before

- Reasoning might satisfy a constraint
  - For example, we infer that required property exists
Summary - Things that matter

- Expressivity of constraints should be equivalent to SPARQL
- There should be a concise syntax that captures most common use cases for constraints
- There should be a mapping from the (one or more) constraint syntax(es) to SPARQL
- Should be possible to fall back to SPARQL syntax when necessary
- Reasoning and RDF validation should work together as in SPARQL entailment regimes