RDF-star and SPARQL-star

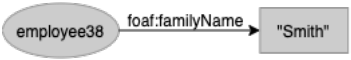
Prof. Dr. Ricardo Usbeck

https://github.com/semantic-systems/rdf-star-tutorial

Overview

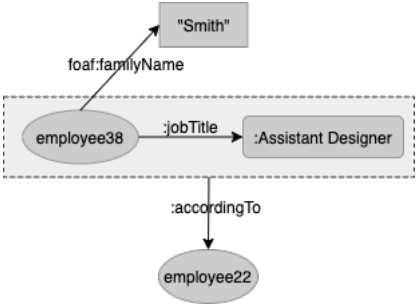
Background and Motivation

- The RDF data model allows you to state world facts as three-part (subject, predicate, object) triples.
  - Predicate of a triple is a property specified with an IRI
  - Subject of a triple and object can each be an IRI referencing any entity, and the object can also be a literal value, dates, numbers, or boolean values
- Example: "employee38 has the familyName Smith" $\rightarrow$ (employee38, familyName, "Smith")



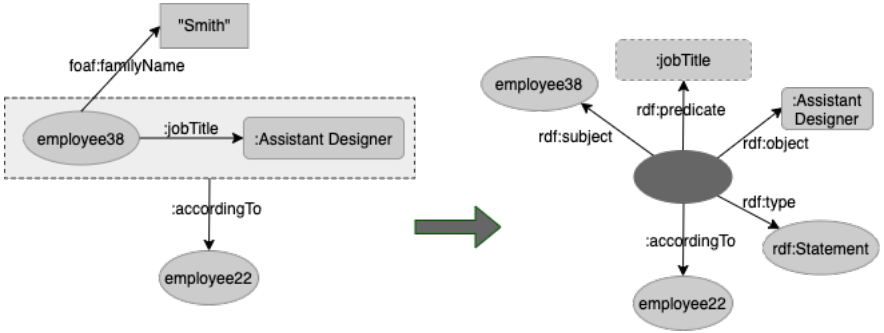
Background and Motivation

- Sometimes, we want the subject or object of a triple to refer to another triple
- Example: "according to employee22, employee38 has a jobTitle of Assistant Designer"



Background and Motivation

- Existing approaches:
  - **Standard Reification** (RDF 1.0, 1999)

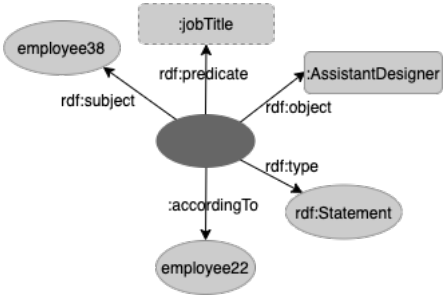


Background and Motivation

- Existing approaches:
  - **Standard Reification** (RDF 1.0, 1999)

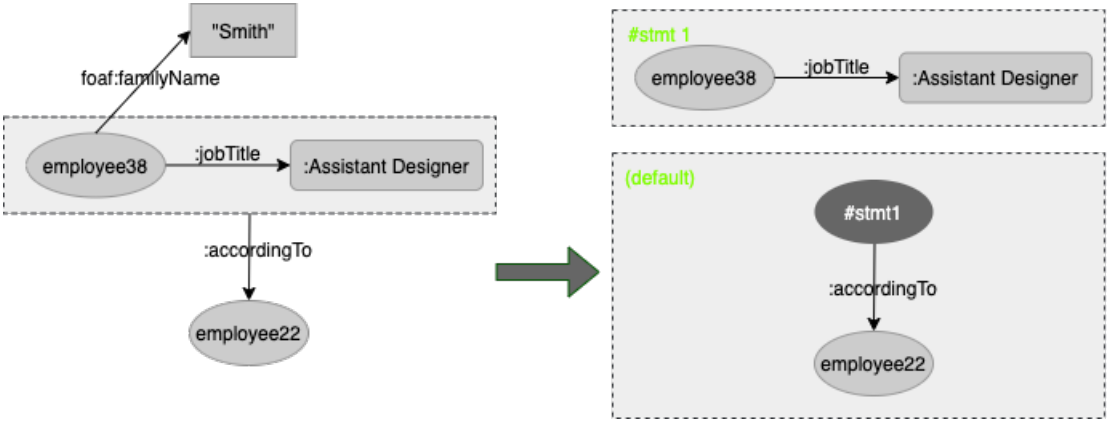
Who has a JobTitle of Assistant Designer according to whom?

```
PREFIX ...
SELECT ?who ?whom WHERE {
 ?claim rdf:type rdf:Statement .
 ?claim rdf:subject ?who .
 ?claim rdf:predicate :jobTitle .
 ?claim rdf:object :AssistantDesigner .
 ?claim :accordingTo ?whom .
}
```



Background and Motivation

- Existing approaches:
  - Standard Reification (RDF 1.0, 1999)
  - **Single-triple Named Graphs** (Carroll et al., 2005; RDF 1.1, 2014)

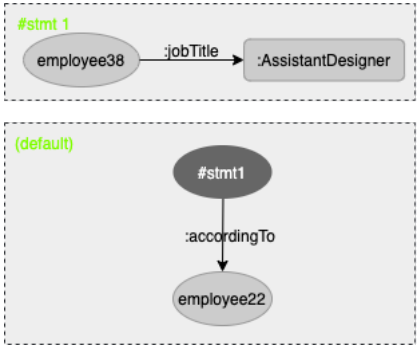


Background and Motivation

- Existing approaches:
  - Standard Reification (RDF 1.0, 1999)
  - **Single-triple Named Graphs** (Carroll et al., 2005; RDF 1.1, 2014)

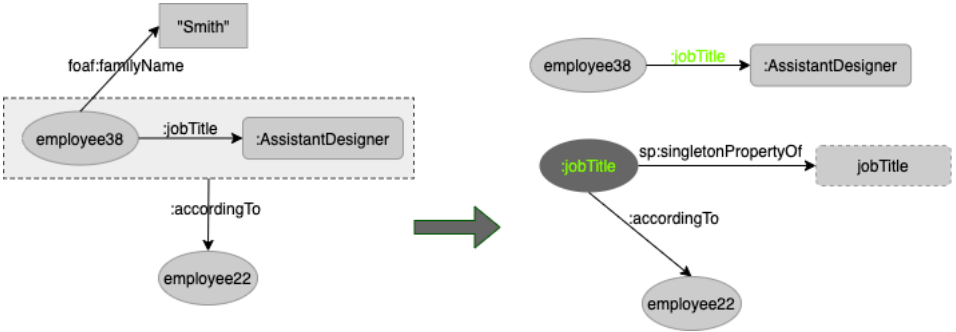
Who has a JobTitle of Assistant Designer according to whom?

```
PREFIX ...
SELECT ?who ?whom WHERE {
 GRAPH ?claim {
  ?who :jobTitle :AssistantDesigner .
 }
 ?claim :accordingTo ?whom .
}
```



Background and Motivation

- Existing approaches:
  - Standard Reification (RDF 1.0, 1999)
  - Single-triple Named Graphs (Carroll et al., 2005; RDF 1.1, 2014)
  - **Singleton Properties** (Nguyen et al., 2014)

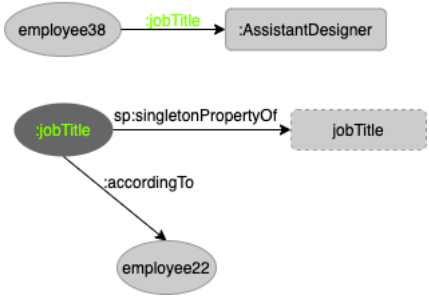


Background and Motivation

- Existing approaches:
  - Standard Reification (RDF 1.0, 1999)
  - Single-triple Named Graphs (Carroll et al., 2005; RDF 1.1, 2014)
  - **Singleton Properties** (Nguyen et al., 2014)

Who has a JobTitle of Assistant Designer according to whom?

```
PREFIX ...
SELECT ?who ?whom WHERE {
    ?who ?claim :AssistantDesigner .
    ?claim sp:singletonPropertyOf :jobTitle .
    ?claim :accordingTo ?whom .
}
```



Background and Motivation

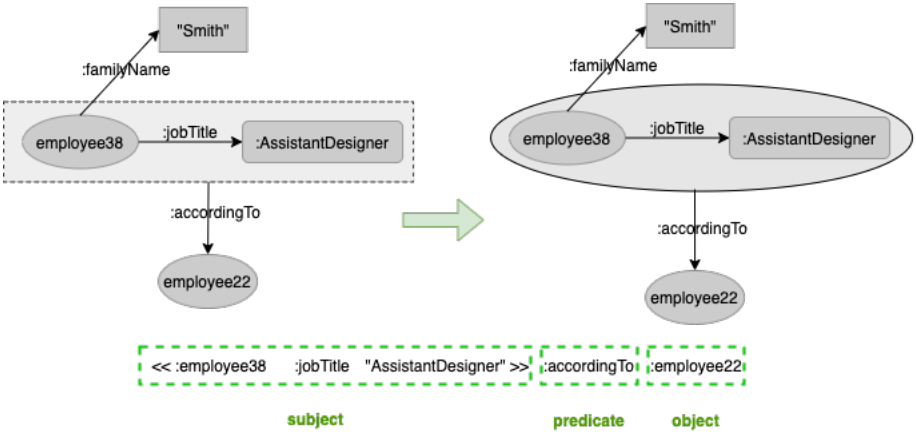
- However, it is **complicated** and **cumbersome** to express metadata about triples in RDF or query it with SPARQL
- Summary of Existing Approaches:
  - Standard Reification
    - ★ Pros: Standard
    - ★ Cons: Verbose; Incomplete/overloaded reified statements
  - Single-triple Named Graphs
    - ★ Pros: Standard
    - ★ Cons: Unspecified semantics; Clutters datasets with "artificial" named graphs
  - Singleton Properties
    - ★ Pros: Relatively concise
    - ★ Cons: Performance issues on many RDF systems

Overview of the RDF-star Approach

- Extension of the RDF conceptual data model and concrete syntax
- Provides a more compact form of reification

Overview of the RDF-star Approach

Basic idea: Nested triples



Overview of the RDF-star Approach

Nested triple patterns

SPARQL to the standard Reification:

```
PREFIX ...
SELECT ?who ?whom WHERE {
 ?claim rdf:type rdf:Statement .
 ?claim rdf:subject ?who .
 ?claim rdf:predicate :jobTitle .
 ?claim rdf:object :AssistantDesigner .
 ?claim :accordingTo ?whom .
}
```

SPARQL-Star over RDF-star is easier to query.

```
PREFIX ...
SELECT ?who ?whom WHERE {
 <<?who :jobTitle "AssistantDesigner">> :accordingTo ?whom .
}
```

Overview of the RDF-star Approach

A brief history

- April 2012, Dagstuhl seminar on Semantic Data Management
- Before 2013, an implementation in Blazegraph ("reification done right")
- June 2014, technical report that defines the RDF*/SPARQL* (Foundations of an Alternative Approach to Reification in RDF)
- Adoption in many systems:
  - Blazegraph, AnzoGraph, Stardog, GraphDB, Neo4j neosemantics
  - Apache Jena, Eclipse RDF4J, RDF.rb, N3.js, EYE
  - YAGO 4 knowledge graph released as a Turtle* file
- March 2019, W3C Workshop on Web Standardization for Graph Data in Berlin
- Community task force as part of the W3C RDF-DEV CG
  - Mixture of implementer, users, and academic researchers
  - Goal: create a spec that captures all aspects of the approach in the form of a CG report, plus a collection of corresponding test suites
  - Some aspects of the approach have changed → new-names: RDF-star, SPARQL-star, etc.

Concepts and Abstract Syntax

RDF-star Data

- **RDF-star graph**: a set of RDF-star triples
  - Any RDF graph is a RDF-star graph
- **RDF-star triple**: a 3-part tuple (subject, predicate, object)
  - Any RDF triple is a RDF-star triple
  - If t and t' are RDF-star triples, s is an IRI or a blank node, p is an IRI, o is an IRI, a blank node or literal, then (t, p, o), (s, p, t) and (t, p, t') are RDF-star triples
- **RDF-star terms**: IRIs, literals, blank nodes and RDF-star triples
- **RDF-star dataset**: a collection of RDF-star graphs, and comprises
  - Exactly one default graph
  - Zero or more named graphs
  - Any RDF dataset is also a RDF-star dataset

Concepts and Abstract Syntax

Asserted Triples vs. Quoted Triples

- **Asserted triple**: RDF-star triple used as the subject or object of another RDF-star triple, also called **embedded triples**.

:employee38 :familyName "Smith"

- **Quoted triple**: RDF-star triple that is an element of a RDF-star graph (and they can be recursive)

« :employee38 :jobTitle "AssistantDesigner" »

Note: A quoted triple does not imply that it also exists as an asserted triple.
Note on the note: An asserted triple (e.g. via annotation on the next slide) cannot be cancelled.

RDF-star Concrete Syntaxes

Turtle-star

- An extension of the Turtle format for representing RDF-star graphs
- Replaces the production rules in the original grammar
- **Grammar**:
  - objectList ::= object **annotation? (',' object annotation?)***
  - subject ::= iri | BlankNode | collection | **quotedTriple**
  - object ::= iri | BlankNode | collection | blankNodePropertyList | literal | **quotedTriple**
  - **quotedTriple** ::= '«' **qtSubject verb qtObject** '»'
  - **qtSubject** ::= **iri | BlankNode | quotedTriple**
  - **qtObject** ::= **iri | BlankNode | literal | quotedTriple**
  - **annoation** ::= '{|' **predicateObjectList** '|}'

RDF-star Concrete Syntaxes

Turtle-star: A simple example

```
PREFIX : <http://www.example.org/>
:employee38 :familyName "Smith" .
<<:employee38 :jobTitle "AssistantDesigner">> :accordingTo :employee22 .
```

versus

```
PREFIX : <http://www.example.org/>
:employee38 :familyName "Smith" .
:employee38 :jobTitle "AssistantDesigner" .
<<:employee38 :jobTitle AssistantDesigner">> :accordingTo :employee22 .
```

RDF-star Concrete Syntaxes

Turtle-star: Annotation Syntax

```
PREFIX : <http://www.example.org/>
:employee38 :jobTitle "AssistantDesigner" {| :accordingTo :employee22 |}.
```

equals

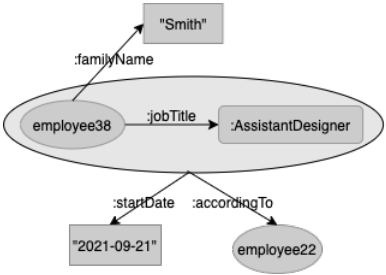
```
PREFIX : <http://www.example.org/>
<<:employee38 :jobTitle "AssistantDesigner">> :accordingTo :employee22.
:employee38 :jobTitle "AssistantDesigner".
```

- Annotation syntax does not appear in the RDF-star abstract data model $\rightarrow$ only a syntactic shortcut
- RDF-star abstract data model does not distinguish how the triples were written

RDF-star Concrete Syntaxes

Turtle-star: A more complex example

```
PREFIX : <http://www.example.org/>
:employee38 :familyName "Smith" .
:jobTitle   "AssistantDesigner" {| :accordingTo :employee22;
                                   :startDate "2021-09-21"^^xsd:date |}.
```



RDF-star Concrete Syntaxes

N-Triples-star

- A minimal extension of the N-Triples format allowing a subject or an object of a RDF-star triple to be a quoted triple.
- No annotation syntax
- **Grammar**:
  - subject ::= IRIREF | BLANK_NODE_LABEL | **quotedTriple**
  - object ::= IRIREF | BLANK_NODE_LABEL | literal | **quotedTriple**
  - **quotedTriple** ::= "«" **subject** **predicate** **object** "»"

RDF-star Concrete Syntaxes

TriG-star

- Minimal extension of the TriG format
- TriG-star document defines a RDF-star dataset, composed of a single default graph and zero or more named graphs, all of which are RDF-star graphs
- **Grammar**:
  - triplesOrGraph ::= labelOrSubject **(** wrappedGraph **|** predicateObjectList **'.')** **|** **quotedTriple** **predicateObjectList '.'**
  - objectList ::= object **annotation?** **(** ',' object **annotation?** **)***
  - subject ::= iri **|** BlankNode **|** collection **|** **quotedTriple**
  - object ::= iri **|** BlankNode **|** collection **|** blankNodePropertyList **|** literal **|** **quotedTriple**
  - **quotedTriple** ::= '«' **qtSubject** **verb** **qtObject** '»'
  - **qtSubject** ::= **iri** **|** **BlankNode** **|** **quotedTriple**
  - **qtObject** ::= **iri** **|** **BlankNode** **|** **literal** **|** **quotedTriple**
  - **annoation** ::= '{|' **predicateObjectList** '|}'

RDF-star Concrete Syntaxes

Other concrete syntaxes

- N-Quads-star
  - For RDF-star datasets
  - N-Triples-star + optional graph name
- JSON-LD-star
  - https://json-ld.github.io/json-ld-star/

SPARQL-star Query Language

Definitions

- A **SPARQL-star triple pattern** is a 3-tuple defined recursively as follows:
  - Every SPARQL triple pattern is a SPARQL-star triple pattern
  - If t and t' are SPARQL-star triple patterns, x is an RDF term or a query variable, and p is an IRI or a query variable, then (t, p, x), (x, p, t), and (t, p, t') are SPARQL-star triple patterns
- A **SPARQL-star basic graph pattern** (BGP-star) is a set of SPARQL-star triple patterns
- A **SPARQL-star property path pattern** is a 3-tuple (s, p, o) where
  - s is either a RDF term, a query variable, or a SPARQL-star triple pattern
  - p is a property path expression, and
  - o is either a RDF term, a query variable, or a SPARQL-star triple pattern
- A **SPARQL-star solution mapping** μ is a partial function from the set of all query variables to the set of all RDF-star terms. The domain of μ is the set of query variable for which μ is defined

SPARQL-star Query Language

Grammar

SPARQL-star is defined to follow the same grammar as SPARQL 1.1, except for the EBNF productions (not complete) specified below.

- Object ::= GraphNode **AnnotationPattern?**
- ObjectPath ::= GraphNodePath **AnnotationPatternPath?**
- GraphNode ::= **VarOrTermOrEmbTP** | TriplesNode
- GraphNodePath ::= **VarOrTermOrEmbTP** | TriplesNodePath
- **EmbTP** ::= '«' **EmbSubjectOrObject** **Verb** **EmbSubjectOrObject** '»'
- **EmbTriple** ::= '«' **DataValueTerm** (iri | 'a') **DataValueTerm** '»'
- **VarOrTermOrEmbTP** ::= **Var** | **GraphTerm** | **EmbTP**

SPARQL-star Query Language

Translation to the Algebra (1/7)

- SPARQL specification defines a process based on the SPARQL grammar, to convert graph patterns and solution modifiers in a SPARQL query string into a SPARQL algebra expression

$\Rightarrow$ Must be adjusted to the extended grammar

- Here: Only discussion of steps which require adjustment

SPARQL-star Query Language

Translation to the Algebra (2/7)

- **Expand variable scope**
  - A variable is in-scope of a BGQ-star B if the variable occurs in B, which includes an occurence in any embedded triple pattern in B (independent of the level of nesting)
  - A variable is in-scope of a property path pattern if the variable occurs in that pattern, which includes an occurence in any embedded triple pattern in the pattern (independent of the level of nesting)

SPARQL-star Query Language

Translation to the Algebra (3/7)

- **Expand Syntax Forms**
  - Annotation patterns **MUST** be replaced by additional SPARQL-star triple pattern that have the annotated triple pattern as an embedded triple pattern in their subject position

```
?x :p :o1 {| :ap1 ?y;
             :ap2 ?z |} ,
      :o2 {| :ap3 ?z |} .
?x :p :o3 {| :ap4/:ap5 ?w } .
```

⇒must be replaced by

```
?x :p :o1, o2 .
<<?x :p :o1>> :ap1 ?y ;
              :ap2 ?z .
<<?x :p :o2>> :ap3 ?z .
?x :p :o3 .
<<?x :p :o3>> :ap4/:ap5 ?w .
```

SPARQL-star Query Language

Translation to the Algebra (4/7)

- **Expand Syntax Forms**
  - Abbreviations for triple patterns with embedded triple patterns **MUST** be expanded as if each embedded triple pattern was a variable (or a RDF-term).

```
<<?c a owl:Class>> dct:source ?src ;
                   :entailing <<?c a rdfs:Class>> .
```

$\Rightarrow$ must be expanded to

```
<<?c a owl:Class>> dct:source ?src .
<<?c a owl:Class>> :entailing <<?c a rdfs:Class>> .
```

SPARQL-star Query Language

Translation to the Algebra (5/7)

- **Expand Syntax Forms**
  - Abbreviations for IRIs in all embedded triple patterns MUST be expanded.

```
<<?c a rdfs:Class>>
```

$\Rightarrow$ must be expanded to

```
<< ?c <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                <http://www.w3.org/2000/01/rdf-schema#Class>>>
```

SPARQL-star Query Language

Translation to the Algebra (6/7)

- **Translate Property Path Patterns**
  - Let *X P Y* be a string where *X* and *Y* may be a RDF term, or an embedded triple pattern, respectively, and *P* is a property path expression (see also `https://www.w3.org/TR/sparql11-query/#sparqlTranslatePathPatterns`)
  - The string *X P Y* is translated to the algebra expression Path(X', *P*, Y') as the result of calling a function named Lift for *X* and *Y*, respectively

Lift

For some input string *Z* that can be a RDF term, a variable, or an embedded triple pattern, the function **Lift** is defined recursively as follows:

1. If *Z* is an embedded triple pattern «S, P, O» then return the SPARQL-star triple pattern (Lift(S), P, Lift(O));
2. Otherwise, return *Z*.

SPARQL-star Query Language

Translation to the Algebra (7/7)

- **Translate Basic Graph Patterns**
  - Triple patterns in the extended syntax may have an embedded triple pattern in their subject position or in their object position (or both).
  - To ensure that every result of this step is a BGP-star, before adding a triple pattern to its corresponding collection, its subject and object **MUST** be replaced by the result of calling function Lift for the subject and the object, respectively

SPARQL-star Query Language

New Built-In Function and Operator Definitions

- **TRIPLE**

  RDF-star triple **TRIPLE** (RDF-star term term1, RDF-star term term2, RDF-star term term3)
- **SUBJECT**

  RDF-star term **SUBJECT** (RDF-star triple triple)
- **PREDICATE**

  RDF-star term **PREDICATE** (RDF-star triple triple)
- **OBJECT**

  RDF-star term **OBJECT** (RDF-star triple triple)
- **isTRIPLE**

  xsd:boolean **isTRIPLE** (RDF-star term term)

SPARQL-star Query Language

New Built-In Function and Operator Definitions

Examples:

```
SELECT ?cartoon ?y ?o
WHERE {
    ?cartoon :depicts << :bob :dreamingOf ?y>>
    FILTER (isTriple(?y))
    BIND (Object(?y) AS ?o)
}
```

```
SELECT ?cartoon ?t
WHERE {
    ?cartoon :depicts << ?x :dreamingOf ?y>>
    BIND (TRIPLE (?x,:state,:sleeping) AS ?t)
}
```

SPARQL-star Query Language

New Built-In Function and Operator Definitions

- **Embedded Triple Expression**

  RDF-star triple « (RDF-star term term1, RDF-star term term2, RDF-star term term3 »
- **sameTerm**

  xsd:boolean **sameTerm** (term, term)
- **sparql-compare**

  xsd:boolean **sparql-compare** (RDF-star term, RDF-star term)
- **RDFterm-equal**

  The function is the default dispatch for the = operator.

Definition: sparql-compare

- If neither A nor B is an RDF-star triple term, compare by SPARQL 1.1 operators <, =, >,) and (, return the comparison value (-1, 0, +1) or throw an error as defined by SPARQL 1.1.
- If either A or B is an RDF-star triple term, and the other is not an RDF-star triple term, then error.
- If sparql-compare(SUBJECT(A), SUBJECT(B)) != 0, then return this value.
- If sparql-compare(PREDICATE(A), PREDICATE(B)) != 0, then return this value.
- Return sparql-compare(OBJECT(A), OBJECT(B))

SPARQL-star Query Language

Function and Operator Definitions

- **Operator Mappings**

SPARQL Binary Operators (SPARQL-star)

Operator	Type(A)	Type(B)	Evaluation	Result type
SPARQL-star Tests				
A = B	RDF triple term	RDF triple term	op:numeric-equal(sparql-compare(A, B), 0)	xsd:boolean
A != B	RDF triple term	RDF triple term	fn:not(op:numeric-equal(sparql-compare(A, B), 0))	xsd:boolean
A < B	RDF triple term	RDF triple term	op:numeric-equal(sparql-compare(A, B), -1)	xsd:boolean
A <= B	RDF triple term	RDF triple term	fn:not(op:numeric-equal(sparql-compare(A, B), 1))	xsd:boolean
A > B	RDF triple term	RDF triple term	op:numeric-equal(sparql-compare(A, B), 1)	xsd:boolean
A >= B	RDF triple term	RDF triple term	fn:not(op:numeric-equal(sparql-compare(A, B), -1))	xsd:boolean

Figure: Source: `https://w3c.github.io/rdf-star/cg-spec/2021-07-01.html`

- **Triple term with ORDER BY**
  - (Lowest) no value assigned to the variable or expression in this solution.
  - Blank nodes
  - IRIs
  - RDF literals
  - RDF-star triple terms

SPARQL-star Query Language

Query Result Formats

SPARQL-star Query Results JSON Format Consider the following RDF term, an embedded triple in Turtle-star syntax:

```
<< <http://example.org/alice> <http://example.org/name> "Alice" >>}}
```

This term is represented in JSON as follows:

```
{"type": "triple",
 "value": {
   "subject": {
       "type": "uri",
       "value" "http://example.org/alice"
   },
   "predicate": {
       "type": "uri",
       "value" "http://example.org/name"
   },
"object": {
"type": "literal",
"value" "Alice",
"datatype": "http://www.w3.org/2001/XMLSchema#string"},
}}
```

SPARQL-star Query Language

Query Result Formats

SPARQL-star Query Results XML Format Consider the following RDF term, an embedded triple in Turtle-star syntax:

```
<< <http://example.org/alice> <http://example.org/name> "Alice" >>}}
```

This term is represented in XML as follows:

```
<triple>
    <subject>
        <uri>http://example.org/alice</uri>
    </subject>
    <predicate>
        <uri>http://example.org/name</uri>
    </predicate>
    <object>
        <literal datatype='http://www.w3.org/2001/XMLSchema#string'>Alice</literal>
    </object>
</triple>
```

SPARQL-star Query Language

SPARQL-star Update

INSERT DATA

```
PREFIX : <http://www.example.org/>
INSERT DATA {
    :alice :claims << :bob :age 23 >> .
}
```

```
PREFIX : <http://www.example.org/>
INSERT DATA {
    :bob :age 23 .
    :alice :claims << :bob :age 23 >> .
}
```

DELETE DATA

```
PREFIX : <http://www.example.org/>
DELETE DATA {
    :alice :claims << :bob :age 23 >> .
}
```

```
PREFIX : <http://www.example.org/>
DELETE DATA {
    :bob :age 23 .
}
```

SPARQL-star Query Language

SPARQL-star Update

DELETE/INSERT

```
PREFIX : <http://www.example.org/>
DELETE { :alice ?pp <<?s ?p ?o>> . }
INSERT { :carol ?pp <<?s ?p ?o>> . }
WHERE {  :alice ?pp <<?s ?p ?o>> . }
```

```
PREFIX : <http://www.example.org/>
DELETE { :alice ?pp <<?s ?p ?o>> . }
INSERT { :carol ?pp <<?s ?p ?o>> .  ?s ?p ?o .}
WHERE {  :alice ?pp <<?s ?p ?o>> . }
```

SPARQL-star Query Language

SPARQL-star Update

DELETE/INSERT

```
PREFIX : <http://www.example.org/>
INSERT {
    GRAPH :graph2 { ?s ?p ?o }
}
WHERE {
    { <<?s ?p ?o>> ?pp ?oo }
    UNION
    { ?ss ?pp <<?s ?p ?o>> }
}
```

Use Cases

- Use Cases for justification are collected
  `https://w3c.github.io/rdf-star/UCR/rdf-star-ucr.html`
- Still an active field of discussion
  `https://lists.w3.org/Archives/Public/public-rdf-star/2021Dec/0001.html`
- (Strongly disputed) Use case by Amazon `https://lists.w3.org/Archives/Public/public-rdf-star/2021Dec/att-0001/rdf-star-neptune-use-cases-20211202.pdf`
  $\rightarrow$ We are still not able to model every real-world use case with satisfaction

Summary

Outlook

- 4th community report underway
  `https://w3c.github.io/rdf-star/cg-spec/editors_draft.html`
- SHACL-star as another extension
- W3C Recommendation Track for the Community Group

Summary

References

- Olaf Hartig, Pierre-Antoine, Gregg Kellogg, and Andy Seaborne. RDF-star and SPARQL-star, Draft Community Group Report, https://w3c.github.io/rdf-star/cg-spec/2021-07-01.html, 01 July 2021.
- Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, https://www.w3.org/TR/rdf11-concepts/, W3C Recommendation, 25 February 2014.
- Steve Harris, Andy Seaborne. SPARQL 1.1 Query Language, https://www.w3.org/TR/sparql11-query/. W3C Recommendation, 21 March 2013.
- https://github.com/semantic-systems/rdf-star-tutorial Simple tutorial
- http://www.lotico.com/index.php/Metadata_for_RDF_Statements:_The_RDF-star_Approach Video lecture by Olaf Hartig and Pierre-Antoine Champin