Reasoning

Make implicit information explicit

Taught by:

Jaroslav Pullmann
Solutions Architect

Learning Objectives

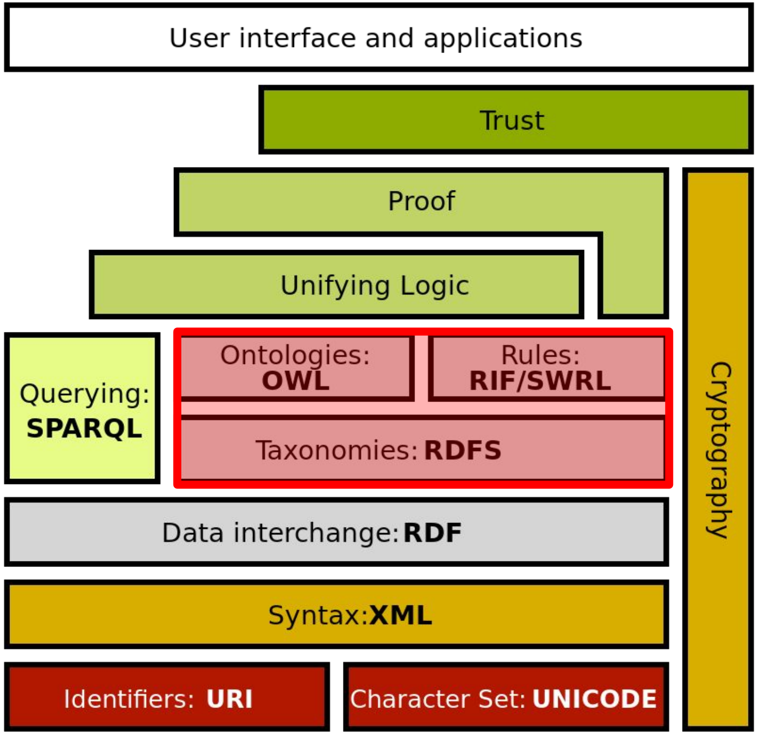
- Become familiar with modeling standards
- Learn about the means to unveil implicit information
- Understand the benefits of logical reasoning

Overview

- Derive new facts (inferences) from the existing, asserted statements
    - Leverage definitions (axioms) of the domain (schema, ontology)
    - Perform logical inference based on proven formalism (Description Logic)
- Make implicit information explicit
    - Infer new facts, e.g. classify resources or compute values
- Reshape and align data
    - Create views and links between nodes
- Derive explainable results
    - Compare to statistical inference
- Verify the correctness of domain model

Relevant Standards

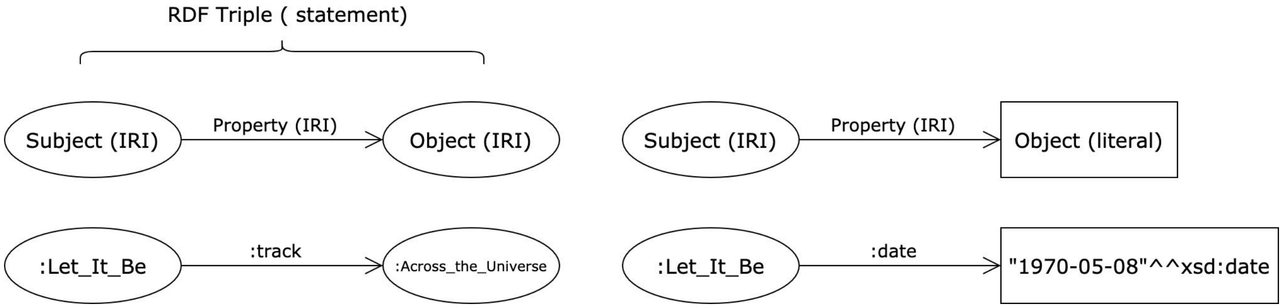
- Multiple layers of standards involved
- Each adding expressivity to reasoning
  - RDF - Instance data (facts)
  - RDFS - Class and property hierarchies
  - OWL - Expressive class definitions
  - SWRL - Rules and functions



Source: https://en.wikipedia.org/wiki/Semantic_Web_Stack

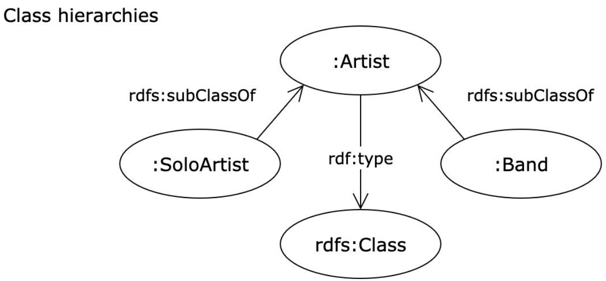
Resource Description Framework (RDF)

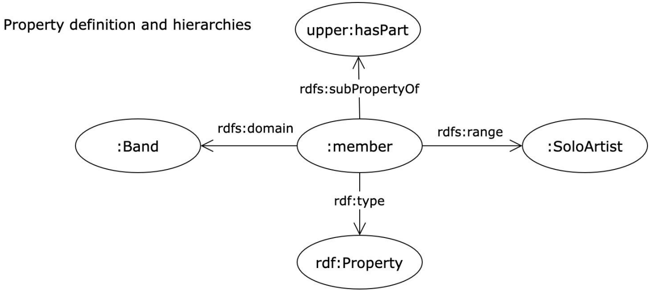
- Basic vocabulary for expressing RDF statements
  - Triples of “subject”, “predicate”, and “object“
  - Triple arguments are IRIs (depicted as ellipses) or literals (rectangles)
- Properties (`rdf:Property`) as a distinguished set of IRIs (attribute or relationship)
  - `rdf:type` - links an instance to corresponding class (`rdfs:Class`)
  - `rdf:value` - standard value attribute of structured (blank) nodes



RDF Schema

- Data modeling vocabulary for RDF (extension of RDF vocabulary)
- Classes (e.g., `rdfs:Class`, `rdfs:Literal`) and class hierarchies (`rdfs:subClassOf`)
  - Set of instances (resources) with common characteristics
- Datatypes (`rdfs:Datatype`)
  - Set of literals of corresponding type (XML Schema or custom)
- Properties (`rdf:Property`) and property hierarchies (`rdfs:subPropertyOf`)
  - Range of applicable classes (`rdfs:domain`) and values (`rdfs:range`)





RDF Schema / Hierarchy Inference

- Match classes, properties down the hierarchy via `rdfs:subClassOf` and `rdfs:subPropertyOf`
- Hierarchy traversal with SPARQL property path: `rdf:type/rdfs:subClassOf*`

RDFS Schema

```
:Artist a rdfs:Class .
:Band a rdfs:Class ;
      rdfs:subClassOf :Artist .
:Person a rdfs:Class .
:SoloArtist a rdfs:Class ;
  rdfs:subClassOf :Artist, :Person .
```

RDF Instance data

```
:The_Beatles rdf:type :Band .
:The_Beatles :member :John_Lennon .
:John_Lennon rdf:type :SoloArtist .
```

SPARQL Query examples

```
select * { ?artist rdf:type :Artist }

select * { ?artist
  rdf:type/rdfs:subClassOf* :Artist }
```

artist
:The_Beatles
:John_Lennon

```
select * { ?artist rdf:type :Person }
```

a
:John_Lennon

RDF Schema / Domain & Range Inference

- `rdfs:domain` / `rdfs:range` axioms will infer type of property's subject / object

Example RDFS Inference

```
# Schema
:track a rdf:Property ;
     rdfs:label "track" ;
     rdfs:comment "A song included in an album." ;
     rdfs:domain :Album ;
     rdfs:range :Song .

# Asserted triple
:Let_It_Be :track :Across_the_Universe .

# Inferred triples
:Let_It_Be rdf:type :Album .
:Across_the_Universe rdf:type :Song .
```

Web Ontology Language (OWL)

- Logic-based language for formal description and machine reasoning about RDF resources
- Distinction of relations (`owl:ObjectProperty`) and attributes (`owl:DatatypeProperty`)
- Characteristics of properties (e.g. uniqueness via `owl:FunctionalProperty`)
- Logical relationships among modelling constructs (classes, datatypes, properties, instances)
  - For example, equivalence, intersection, or disjointness
- Captures the meaning of classes via structural expressions
  - Based on values or cardinality of properties

Example OWL 2 axioms

```
# Equivalence of classes (same concepts)
a:Song owl:equivalentClass b:Song .

# Equivalence of individuals (same instances)
dbr:Ringo_Starr owl:sameAs dbr:Richard_Starkey
```

OWL 2 Class Axioms

- Class is a set of instances with common characteristics
- Relationships between classes are expressed in terms of set theory

- Extension (instance set) of a superclass includes extensions of its subclasses
- Equivalent classes share the same extension, i.e., the same set of instances
- Disjoint classes represent incompatible concepts and have no instances in common
  - Membership in one class excludes membership in the disjoint class

OWL 2 Class Expressions / Restrictions

- In RDF, properties are IRIs with a “global” visibility and definition (`rdfs:domain`, `rdfs:range`)
- OWL 2 Class expressions define class extensions based on local property restrictions
- Instances of `owl:Restriction` refer to constrained property via `owl:onProperty`

OWL 2 Class expression

```
# An Album has at least one track (Song)
[
     rdf:type owl:Restriction ;
     owl:onProperty :track ;
     owl:someValuesFrom :Song
] rdfs:subClassOf :Album
```

- Existential quantification (`owl:someValuesFrom`) requires at least one value of given type
- Universal quantification (`owl:allValuesFrom`) requires all known values be of given type
- Cardinality restrictions (e.g. `owl:cardinality`) restrict the number of distinct property values
  - Note: OWL 2 profiles do *not* support arbitrary cardinatlity restrictions

OWL 2 Class Expressions (2)

- Value restrictions (`owl:hasValue`) require the property to have at least the given value

Value restriction

```
# A rock album has the genre Rock Music
[
      rdf:type owl:Restriction ;
      owl:onProperty :genre ;
      owl:hasValue genre:RockMusic
] rdfs:subClassOf :RockAlbum .
```

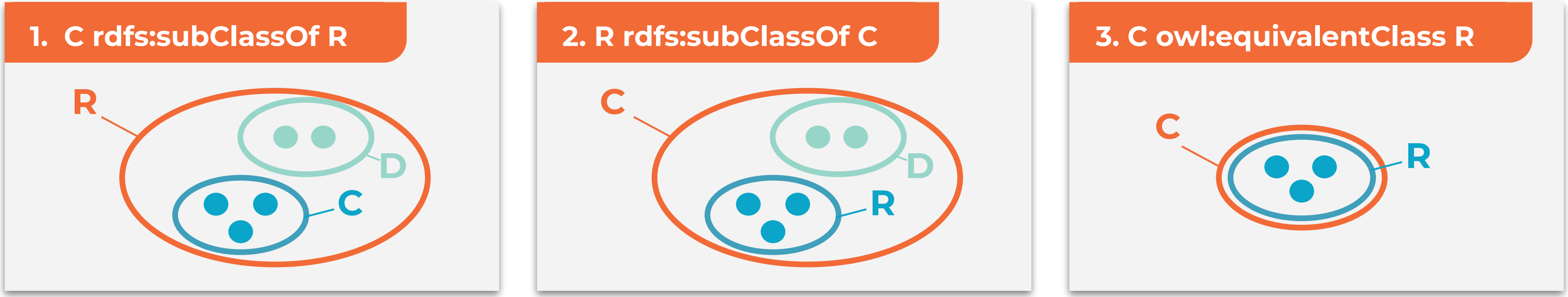
- Enumeration of individuals (`owl:oneOf`) defines a class by an exhaustive listing of instances
  - Any further asserted instances make the database inconsistent

Instance enumeration

```
# Specify Beatles members (class extension) by enumeration
:BeatlesMember
  rdf:type owl:Class ;
  owl:oneOf ( dbr:John_Lennon dbr:Paul_McCartney ... ).
```

OWL 2 Class Definitions

- Use class axioms to define a named class (C) based on a restriction (R)
- Necessary and sufficient condition needs to hold to classify individual (I)



1. **C ⊂ R**: Sufficient condition not met (C can't be distinguished from D)
2. **R ⊂ C**: Necessary and sufficient condition met, I classified as C when matching R
3. **R ≡ C**: Necessary and sufficient condition met, I classified as C when matching R

Use equivalent classes with care. Test whether they have the same meaning in all contexts

OWL 2 Class Definition / Example

- Having genre `:RockMusic` is one of the conditions for an `:Album` to classify as `:RockAlbum`
- Alternative conditions to classify as `:RockAlbum` may exist (independently of genre)

R rdfs:subClassOf C

```
:RockAlbum rdfs:subClassOf :Album .
[
     rdf:type owl:Restriction ;
     owl:onProperty :genre ;
     owl:hasValue :RockMusic
] rdfs:subClassOf :RockAlbum .
```

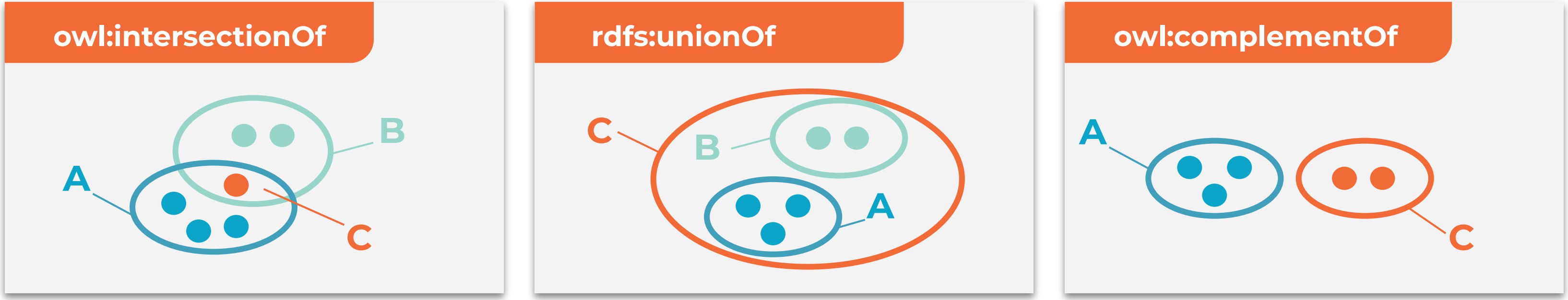
- Having genre `:RockMusic` is exactly the condition for an `:Album` to classify as `:RockAlbum`
- Genre `:RockMusic` is inferred for any `:RockAlbum`

C owl:equivalentClass R

```
:RockAlbum rdfs:subClassOf :Album .
[ ...
] owl:equivalentClass :RockAlbum .
```

OWL 2 Class Expressions / Complex Classes

- Logical class constructors combine class expressions via logical operators `and`, `or`, and `not`



- Intersection (conjunction) of two classes are individuals that are instances of both classes
  - Equivalent to: `C rdfs:subClassOf A . C rdfs:subClassOf B .`
- Union (disjunction) of two classes consists of instances of at least one of the classes
  - Equivalent to: `A rdfs:subClassOf C . B rdfs:subClassOf C .`
- Complement (negation) of a class are individuals which are not members of that class
  - Equivalent to: `A owl:disjointWith C .`

OWL 2 Class Definition / Example (2)

Logical class constructor

```
# A rock band is a band with at least one rock album
:RockBand owl:equivalentClass
    [ a owl:Class ;
        owl:intersectionOf (
            :Band
            [ a owl:Restriction;
                owl:onProperty :hasAlbum;
                owl:someValuesFrom [
                  a owl:Restriction;
                    owl:onProperty :genre;
                    owl:hasValue :Rock
                ]
            ]
        )
    ] .
```

OWL 2 Property Axioms / Overview

- OWL differentiates attributes (`owl:DatatypeProperty`) and relations (`owl:ObjectProperty`)
- Axioms mutually relate properties or describe characteristics of a single property
- Axioms for relations and attributes
  - `rdfs:subPropertyOf`
  - `rdfs:domain`
  - `rdfs:range`
  - `owl:equivalentProperty`
  - `owl:propertyDisjointWith`
- Characteristics of attributes and relations
  - `owl:FunctionalProperty`
- Axioms for relations only
  - `owl:propertyChainAxiom`
  - `owl:inverseOf`
- Characteristics of relations
  - `owl:InverseFunctionalProperty`
  - `owl:ReflexiveProperty`
  - `owl:IrreflexiveProperty`
  - `owl:SymmetricProperty`
  - `owl:AsymmetricProperty`

Equivalent and disjoint Properties

- Equivalent properties are aliases that may be used interchangeably
  - Connect the same sets of individuals
- Consider extending a property (`rdfs:subPropertyOf`) if it does not share all implications

```
:author owl:equivalentProperty :writer .     # Definition
:Queenie_Eye :writer :Paul_McCartney   .     # Assertion
:Queenie_Eye :author :Paul_McCartney   .     # Inference
```

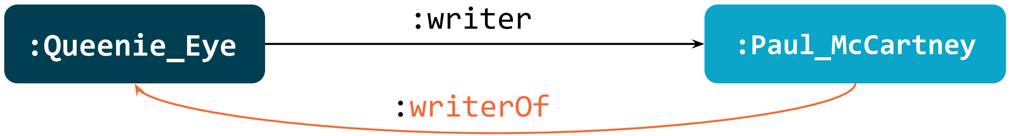
- Disjoint properties must not be used interchangeably
  - Connect distinct sets of individuals (individuals assumed to be mutually different)

```
:father owl:propertyDisjointWith :uncle .    # Definition
```

- As disjointness primarily targets validation we recommend using SHACL (`sh:disjoint`)

Inverse Properties

- RDF is a directed labeled graph, properties link subjects to objects of statements (triples)
- Properties are directional (`:writer` links Song to Songwriter)



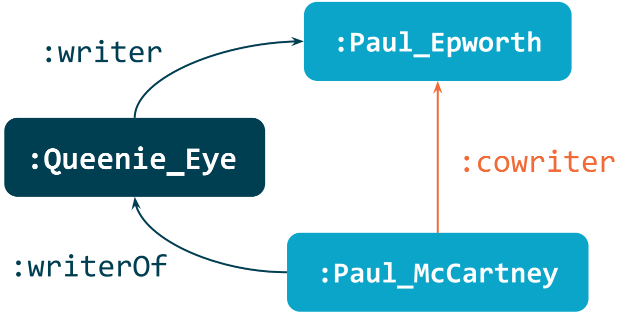
- Inverse property flips the direction of a property
- `:writerOf` links Songwriter to Song (i.e., from object to subject of the inverted property)

```
:writerOf owl:inverseOf :writer .           # Definition
:Queenie_Eye :writer :Paul_McCartney .      # Assertion
:Paul_McCartney :writerOf :Queenie_Eye .    # Inference
```

```
select ?song where { :Paul_McCartney ^:writer ?song }# SPARQL, inverse property path
```

Property Chain

- Infer a new relation between two nodes connected via a property chain (enumerated path)
- There may be arbitrary number of relations in the chain (asserted and inferred)

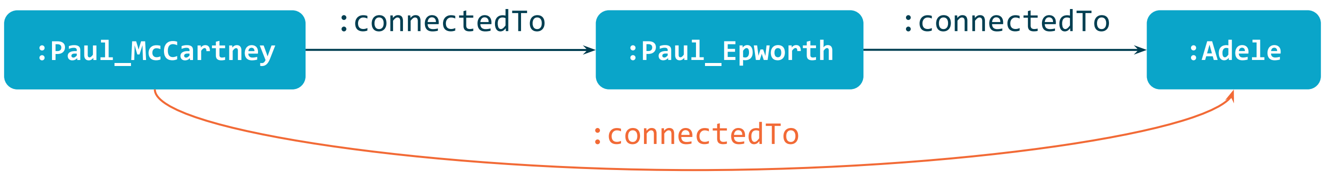



```
:cowriter owl:propertyChainAxiom ( :writerOf :writer ) # Definition
:Paul_McCartney :writerOf :Queenie_Eye. # Assertions
:Queenie_Eye :writer :Paul_Epworth .
:Paul_McCartney :cowriter :Paul_Epworth . # Inference
```

```
select ?cowriter where { :Paul_McCartney :writerOf / :writer ?cowriter } # SPARQL
```

Transitive Properties

- Infer a new edge between two nodes connected via a path of the same property
- Special case of a property chain (path of arbitrary length)




```
:connectedTo a owl:TransitiveProperty .         # Definition
:Paul_McCartney :connectedTo :Paul_Epworth .  # Assertions
:Paul_Epworth :connectedTo :Adele .
:Paul_McCartney :connectedTo :Adele .           # Inference
```

```
select ?contact where {:Paul_McCartney :connectedTo+ ?contact }#SPARQL property path
```

Reflexive / Irreflexive Properties

- Reflexive properties relate an individual to itself (subject and object is the exact same node)
- Used, e.g., with mereological (containment) relations (anything is trivially part of itself)


```
:partOf a owl:ObjectProperty , owl:ReflexiveProperty . # Definition
```

- In contrast, irreflexive properties are known to connect distinct individuals only
- The same individual cannot be related to itself via an irreflexive property

```
:childOf a owl:ObjectProperty , owl:IrreflexiveProperty . # Definition
```

- As with `owl:propertyDisjointWith` axiom we recommend using SHACL (`sh:disjoint`)

Symmetric Properties

- Symmetric properties are bidirectional (direction is irrelevant)
  - `:connectedTo` relationship will hold in both directions

- A symmetric property is its own inverse

```
:connectedTo a owl:ObjectProperty , owl:SymmetricProperty . # Definition
:Adele :connectedTo :Paul_McCartney . # Assertion
:Paul_McCartney :connectedTo :Adele . # Inference
```

```
select ?p1 ?p2 where { ?p1 :connectedTo | ^:connectedTo ?p2 } # SPARQL, alt. path
```

Functional Properties / Unique Values

- Functional properties reference a value unique per resource, e.g., publication date
- There is at most one outgoing property occurrence (referred triple object)
- For validation of functional values consider SHACL `sh:maxCount` constraint

```
# An album should have only one date
:date rdf:type owl:DatatypeProperty , owl:FunctionalProperty .
:Abbey_Road :date "1969-09-26"^^xsd:date .
```

- Inverse functional properties reference a globally unique value, e.g., ISBN number
- There at most one incoming property occurrence (referring triple subject)
- Consider a combination of `sh:maxCount` and `sh:inversePath` SHACL constraints for validation

```
# A medium (CD, book) is uniquely identified by an International Standard Book Number
:isbn13 rdf:type owl:ObjectProperty , owl:InverseFunctionalProperty .
:Abbey_Road :isbn13 isbn:978-1608199990 .
```

Inverse Functional Property / Validation (1)

OWL / RDF

```
# General schema axioms
:isbn13 rdf:type
      owl:ObjectProperty ,
      owl:InverseFunctionalProperty .

# Asserted instance data
:Abbey_Road  :isbn13  isbn:978-1608199990 .
:Other_Album :isbn13  isbn:978-1608199990 .

# Non-unique Naming Assumption
#
# In RDF/OWL different IRIs do not necessarily refer to
# distinct resources since no coordination of (unique)
# resource naming on global scale is assumed.
# We'll need to explicitly differentiate individuals
# and assert disjointness of "Abbey Road" and "Other Album":

:Abbey_Road owl:differentFrom :Other_Album .
```

OWL Consistency check: Values of `:isbn13` property are stated to uniquely identify a resource (`owl:InverseFunctionalProperty`).

Resources with the same `:isbn13` value could be inferred to be equivalent. Due to non-unique naming assumption we'll need to explicitly differentiate them via `owl:differentFrom` axiom.

Using the DL (Description Logic) reasoning mode Stardog will detect an inconsistency:

```
$stardog reasoning explain --inconsistency music

INFERRED
 ASSERTED :Other_Album :isbn13 isbn:978-1608199990
 ASSERTED :Abbey_Road  :isbn13 isbn:978-1608199990
 ASSERTED :Abbey_Road owl:differentFrom :Other_Album
 ASSERTED :isbn13 a owl:InverseFunctionalProperty
```

Inverse Functional Property / Validation (2)

SPARQL

```
# ISBN is expected to be globally unique,
# report any ISBN reused for distinct albums

select ?isbn (count(distinct ?album) as ?count)  {
    ?album :isbn13 ?isbn
}group by ?isbn having (?count > 1)
```

SHACL

```
# Looking at identifier resources (validation target)
# require at most at most one (inverse) link to an album

:ISBNShape
     a sh:NodeShape ;
     sh:targetObjectsOf :isbn13 ;
     sh:property [
          sh:path [ sh:inversePath :isbn13 ] ;
          sh:maxCount 1 ;
     ] .
```

Unique Values / Keys

- (Complex) instance key
- Set of predicates which values jointly constitute a unique identifier of a class instance
- Defined as a list of predicates (data and object properties) on a class expression

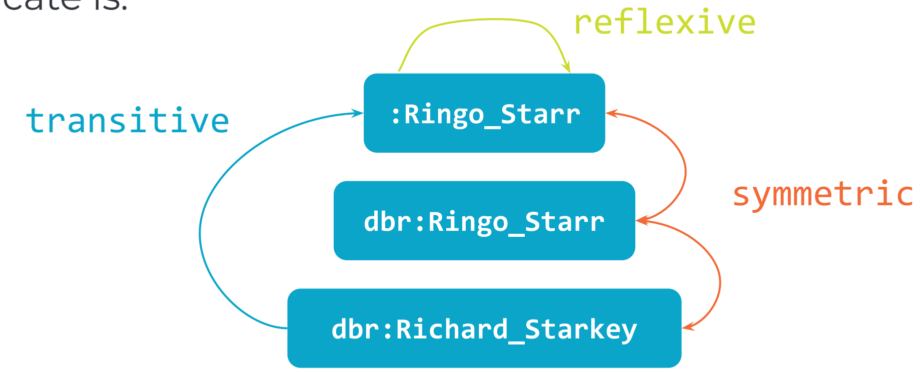
```
# A piece of media should be uniquely identified by an ISBN
:MusicCD rdfs:subClassOf :Medium . :Medium owl:hasKey ( :isbn13 ) .
```

OWL Property Axioms / Alternatives

- Number of RDFS/OWL axioms initially designed for validation (consistency checking)
- Validation axioms:
  - `rdfs:domain`
  - `rdfs:range`
  - `owl:FunctionalProperty`
  - `owl:InverseFunctionalProperty`
  - `owl:IrreflexiveProperty`
  - `owl:ReflexiveProperty`
- Alternative: SHACL
- See OWL - SHACL Comparision
- Axioms recommended for reasoning:
  - `rdfs:subClassOf`
  - `rdfs:subPropertyOf`
  - `owl:inverseOf`
  - `owl:propertyChainAxiom`
  - `owl:TransitiveProperty`
  - `owl:SymmetricProperty`
- Alternative: SPARQL Property Paths

Axioms on Individuals / owl:sameAs

- Declare different resources (IRIs) to represent the same entity
- Nodes will be merged at logical level
- `owl:sameAs` predicate is:




```
:Ringo_Starr owl:sameAs dbr:Ringo_Starr, dbr:Richard_Starkey .
```

owl:sameAs / Canonical IRI

- One canonical individual selected from each `owl:sameAs` equivalence set (aliases)
  - Chosen randomly, but fixed until the set has changed (updates to data or schema)
  - Only the canonical individual (one hit) returned by queries regardless of the set size
- Queries may use any (asserted or inferred) alias, not only the canonical one

```
select * where { dbr:Richard_Starkey ?p ?o } # :Ringo_Starr is the canonical IRI
```

- Query for aliases via `owl:sameAs` predicate

```
select ?otherAliases { dbr:Richard_Starkey owl:sameAs ?otherAliases }
```

owl:sameAs / Implementation

- `owl:sameAs` inferences are computed and indexed eagerly (materialized)
  - `owl:sameAs` index is updated automatically when the database changed
- configuration option `reasoning.sameas` defining the reasoning type:

`OFF` - no inferences, only asserted `owl:sameAs` triples will be included in query results

`ON` - extends asserted `owl:sameAs` predicate by its reflexivity, symmetry and transitivity

`FULL` - extends the above by functional, inverse functional properties, and hasKey axioms

owl:sameAs / Example

sameAs reasoning types

```
# sameAs via transitivity/symmetry, type ON
:Ringo_Starr owl:sameAs dbr:Ringo_Starr, dbr:Richard_Starkey .

:isbn13 a owl:DatatypeProperty , owl:InverseFunctionalProperty ;

# same by inverse functional property, type FULL
:Abbey_Road :isbn13 isbn:978-1608199990 .
:my_album :isbn13   isbn:978-1608199990 .


:producer a owl:ObjectProperty , owl:FunctionalProperty ;
:Album owl:hasKey ( :date :producer ) .

# same via key, type FULL
:Abbey_Road :date "1969-09-26"^^xsd:date ; :producer :George_Martin .
:my_other_album :date "1969-09-26"^^xsd:date ; :producer :George_Martin .
```

Rule-based Reasoning

- Reasoning based on user-defined rules to augment OWL axioms
- IF/THEN constructs with condition and consequence expressed in terms of graph patterns

Stardog Rule Syntax

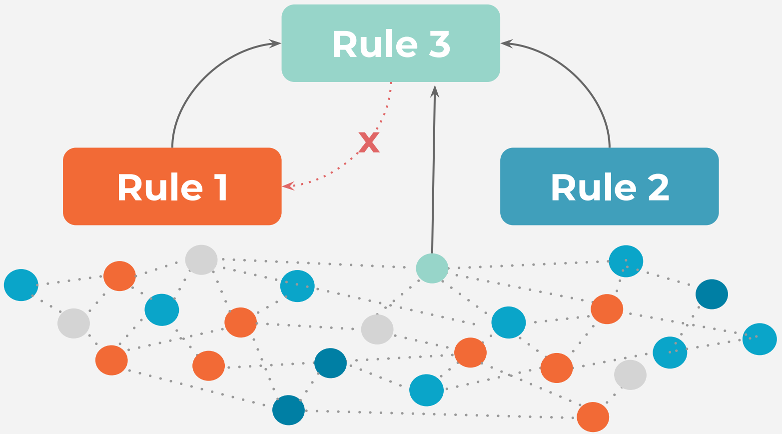
```
# Abstract syntax
IF { condition } THEN { consequence }

# Classifies a band based on the :genre value
IF {
     ?band a :Band .
     ?album a :Album ;
          :artist ?band ;
          :genre genre:Rock .
}
THEN {
   ?band a :RockBand
}
```

Rules-based Reasoning / Benefits (1)

- Rules may build upon each other (without cycles) introducing layers of abstraction
- Rules may maintain a stable data contract thanks to the *inferred* data structure
    - Updates to instance data will not break query graph patterns based on rules

Layers of Abstraction



Rules-based Reasoning / Benefits (2)

- Rules hide complexity of the data and implementation details
  - Classify resources based on their attributes
  - Compute values of inferred properties

Stardog Rule Inference

```
if{
      ?album a :Album ; :track  ?track .
      ?track :length ?length
      filter(?length > 3600) # longer than 1 hour
      bind(round((?length / 3600)) as ?lengthInHours)
}
then{
      # Classify the individual
      ?track a :LongTrack ;
      # Infer computed value
      :lengthInHours ?lengthInHours .
}
```

Schema Management

- Reasoning depends on schema information and rules present in the database
  - These are just RDF triples, loaded like any graph data (e.g., `data add` command)
- "Schema graphs" identified by the database property `reasoning.schema.graphs`
  - Comma-separated list of named graph IRIs, including the special named graphs
  - Used for reasoning with default schema
- Reasoning with multiple, custom schemas (schema multi-tenancy)
  - "Schema": named, custom set of named graphs (selected at query time)
- Rationale
  - Compatibility: various versions of a schema used by (legacy or recent) applications
  - Evolution: different rules and business logic, e.g. threshold value computation by rules
  - Scalability and modularization: partitioning of a large number of axioms and rules

Schema Management (2)

- Schemas defined via the configuration option `reasoning.schemas`
  - Comma-separated collection of `<schema name>=<named graph IRI>` pairs
  - Graphs for the default schema configured via `reasoning.schema.graphs`
- Schema management via CLI:

```
stardog reasoning schema --add musicSchema --graphs :charts :albums -- musicDb
stardog reasoning schema --remove musicSchema -- musicDb
stardog reasoning schema --list musicDb
```

Stardog Reasoning / Commands

- Reasoning is disabled by default, use `-r` (`reasoning=true`) or `--schema` paremeters to activate

CLI

```
# --reasoning has the same effect as --schema default
$ stardog query execute --reasoning music "SELECT ?s { ?s a :RockAlbum } LIMIT 10"

$ stardog query execute --schema musicSchema music "SELECT ?s ..."
```

HTTP API

```
curl -u admin:admin "http://localhost:5820/myDB/query?reasoning=true&query="..."

curl -u admin:admin "http://localhost:5820/myDB/query/reasoning?query=..."
```

Debug Reasoning

- Stardog exposes explanations of an inference
  - Minimum set of statements (asserted and inferred) to logically justify the inference
- Proof tree
  - Explanation of an inference or an inconsistency as a hierarchical structure
  - Multiple assertion nodes are grouped under an inferred node
  - Merges related explanations into a single proof tree (merged proof tree)
  - Alternatives for an explanation are shown with a number id
    - For example `1.1` and `1.2` are both alternative explanations for inference 1

```
$ stardog reasoning explain music ":Abbey_Road a :RockAlbum"
INFERRED :Abbey_Road a :RockAlbum
   ASSERTED (:genre value genre:Rock) rdfs:subClassOf :RockAlbum
   ASSERTED :Abbey_Road :genre genre:Rock
```

Reasoning in Stardog / Query Rewriting

- Reasoning in Stardog is performed at query time via query rewriting:
  - Client queries are augmented with respect to selected reasoning schema and profile
    - Reasoning schema: named set of (domain-specific) model statements (axioms)
    - Reasoning profile: selection of modeling constructs and related inference rules
  - Expanded query is executed against the dataset in a standard manner
  - Scales well even for large datasets (*pay for reasoning that you actually use*)
- In contrast: Inference materialization expands the dataset with respect to a schema/profile
  - Management issues: materialization on each schema update or reasoning profile
  - Resource issues: materialization may be computationally expensive (CPU time)
  - Scaling issues: significant increase of data size, I/O penalty applies to every query
- Exception to query rewriting: `owl:sameAs` reasoning is eagerly materialized

Custom Inference Materialization

- Ingest inference results into database (a dedicated named graph) to improve query times
- Materialization will decouple the production of inferences from querying (no reasoning)

Store inferences

```
# stardog query execute --reasoning store_inferences.rq
DROP SILENT GRAPH <urn:inferences> ;
INSERT {
    GRAPH <urn:inferences> {
         ... # store asserted and inferred results
    }
}
WHERE {
  ...
}
```

Demo

Resources

- OWL 2 Primer
- OWL 2 Quick Reference Guide
- OWL 2 Reference card (PDF)
- Video training on reasoning (by Stardog CTO Evren Sirin)
- Stardog reasoning documentation
- Stardog blog posts on reasoning

Learning Objectives

- Become familiar with modeling standards
- Learn about the means to unveil implicit information
- Understand the benefits of logical reasoning

Thank you