Knowledge Graphs 2022/23: Mock Exam

Maximilian Marx

KBS group, TU Dresden

2023-02-07

Task 1: a–b

RDF Literals I

Let the prefix `xsd` be defined as `http://www.w3.org/2001/XMLSchema#`. Which of the following three RDF literals describe the same value?

1. `"2"^^xsd:integer`
2. `"2"^^xsd:float`
3. `"2.0"^^xsd:decimal`

1. `"2021-03-16T09:20:00"^^xsd:dateTime`
2. `"2021-03-16T09:20:00+01:00"^^xsd:dateTime`
3. `"2021-03-16T08:20:00Z"^^xsd:dateTime`

Task 1: a–b

RDF Literals I

Let the prefix `xsd` be defined as `http://www.w3.org/2001/XMLSchema#`. Which of the following three RDF literals describe the same value?

1. `"2"^^xsd:integer`
2. `"2"^^xsd:float`
3. `"2.0"^^xsd:decimal`

1. `"2021-03-16T09:20:00"^^xsd:dateTime`
2. `"2021-03-16T09:20:00+01:00"^^xsd:dateTime`
3. `"2021-03-16T08:20:00Z"^^xsd:dateTime`

Solution

- `"2"^^xsd:integer` describes the same value as `"2.0"^^xsd:decimal`, since `xsd:integer` is a derived type of `xsd:decimal`

Task 1: a–b

RDF Literals I

Let the prefix `xsd` be defined as `http://www.w3.org/2001/XMLSchema#`. Which of the following three RDF literals describe the same value?

1. `"2"^^xsd:integer`
2. `"2"^^xsd:float`
3. `"2.0"^^xsd:decimal`

1. `"2021-03-16T09:20:00"^^xsd:dateTime`
2. `"2021-03-16T09:20:00+01:00"^^xsd:dateTime`
3. `"2021-03-16T08:20:00Z"^^xsd:dateTime`

Solution

- `"2"^^xsd:integer` describes the same value as `"2.0"^^xsd:decimal`, since `xsd:integer` is a derived type of `xsd:decimal`
- `"2021-03-16T09:20:00+01:00"^^xsd:dateTime` and `"2021-03-16T08:20:00Z"^^xsd:dateTime` describe the same value in different time zones

Task 1: c

RDF Literals II

Define the term “RDF graph”. You may use the terms “IRI”, “blank node”, “literal”, and “Unicode” in your definition.

Task 1: c

RDF Literals II

Define the term “RDF graph”. You may use the terms “IRI”, “blank node”, “literal”, and “Unicode” in your definition.

Solution

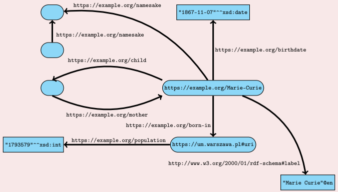
A **triple** is a 3-tuple of **subject**, **predicate**, and **object**, where

- subject is an IRI or a blank node,
- predicate is an IRI, and
- object is an IRI, a blank node, or a literal.

An **RDF graph** is a set of triples.

Task 2

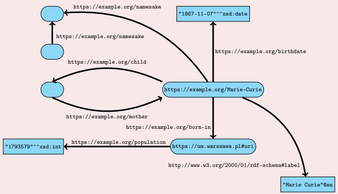
Turtle Serialisation



Use the Turtle format to encode this RDF graph, using the base IRI `https://example.org/` and the prefixes `xsd` (for `http://www.w3.org/2001/XMLSchema#`) and `rdfs` (for `http://www.w3.org/2000/01/rdf-schema#`). Take advantage of syntactic abbreviations wherever possible.

Task 2

Turtle Serialisation



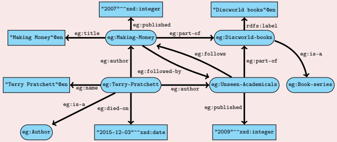
Use the Turtle format to encode this RDF graph, using the base IRI `https://example.org/` and the prefixes `xsd` (for `http://www.w3.org/2001/XMLSchema#`) and `rdfs` (for `http://www.w3.org/2000/01/rdf-schema#`). Take advantage of syntactic abbreviations wherever possible.

Solution

```
<Marie—Curie> <birthdate> "1867—11—07"^^xsd:date ;
              rdfs:label "Marie Curie"@en ;
              <born—in> <https://um.warszawa.pl#uri> ;
              <child> [ <mother> <Marie—Curie> ] ;
              <namesake> _:1 .
_:2 <namesake> _:1 .
<https://um.warszawa.pl#uri> <population> 1793579 .
```

Task 3: a

SPARQL Querying I

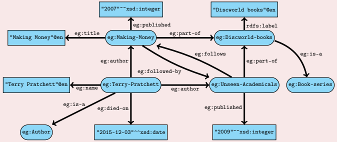


Consider an RDF graph describing authors and book series, using a schema as described by the image to the left (i.e., the image shows just a tiny portion of the whole graph, which contains information on many more book series, authors, and books).

Write a SPARQL query that finds all books belonging to the Discworld series.

Task 3: a

SPARQL Querying I



Consider an RDF graph describing authors and book series, using a schema as described by the image to the left (i.e., the image shows just a tiny portion of the whole graph, which contains information on many more book series, authors, and books).

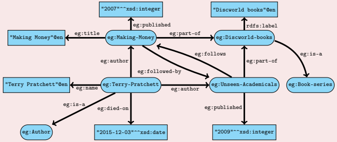
Write a SPARQL query that finds all books belonging to the Discworld series.

Solution

```
SELECT ?book WHERE {
  ?book eg:part—of eg:Discworld—books .
}
```

Task 3: b

SPARQL Querying II

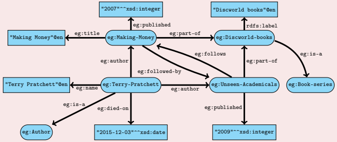


Consider an RDF graph describing authors and book series, using a schema as described by the image to the left (i.e., the image shows just a tiny portion of the whole graph, which contains information on many more book series, authors, and books).

Write a SPARQL query that finds all authors of book series, ordered by the maximal number of books they contributed to any series. Note that multiple authors may contribute books to a series, and that authors may contribute books to multiple series.

Task 3: b

SPARQL Querying II



Consider an RDF graph describing authors and book series, using a schema as described by the image to the left (i.e., the image shows just a tiny portion of the whole graph, which contains information on many more book series, authors, and books).

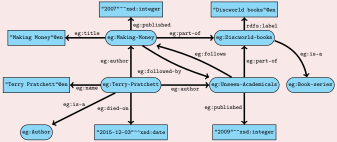
Write a SPARQL query that finds all authors of book series, ordered by the maximal number of books they contributed to any series. Note that multiple authors may contribute books to a series, and that authors may contribute books to multiple series.

Solution

```
SELECT ?author (MAX(?contributions) AS ?books) WHERE {
  { SELECT ?author ?series (COUNT(?book) AS ?contributions) WHERE {
      ?series eg:is—a eg:Book—series .
      ?author eg:author ?book .
      ?book eg:part—of ?series .
  } GROUP BY ?author ?series }
} GROUP BY ?author
ORDER BY DESC(?books)
```

Task 3: c

SPARQL Querying III

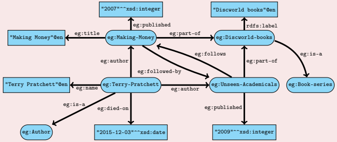


Consider an RDF graph describing authors and book series, using a schema as described by the image to the left (i.e., the image shows just a tiny portion of the whole graph, which contains information on many more book series, authors, and books).

Write a SPARQL query that finds all book series consisting only of books whose authors have died. Note that multiple authors may contribute books to a series.

Task 3: c

SPARQL Querying III



Consider an RDF graph describing authors and book series, using a schema as described by the image to the left (i.e., the image shows just a tiny portion of the whole graph, which contains information on many more book series, authors, and books).

Write a SPARQL query that finds all book series consisting only of books whose authors have died. Note that multiple authors may contribute books to a series.

Solution

```
SELECT ?series WHERE {
  ?series eg:is-a eg:Book-series .
  FILTER NOT EXISTS {
    ?book eg:part-of ?series ; ^eg:author ?author .
    FILTER NOT EXISTS { ?author eg:died-on [] . }
  }
}
```

Task 4: a

RDF Leanness I

Give a formal definition of a **lean RDF graph**. Briefly describe the intuitive significance of this notion.

Task 4: a

RDF Leanness I

Give a formal definition of a **lean RDF graph**. Briefly describe the intuitive significance of this notion.

Solution

An **instance** $\sigma(G)$ of an RDF graph G is obtained by replacing some blank nodes by arbitrary RDF terms. An RDF graph G is lean if it does not have an instance $\sigma(G)$ with $\sigma(G) \subsetneq G$. Intuitively, lean graphs do not have internal redundancy.

Task 4: b

RDF Leanness II

Which of the following RDF graphs is not lean?

1.
```
eg:s   eg:p   eg:o .
_:1    eg:p   _:1  .
```

2.
```
eg:s   eg:p   _:2  .
_:1    eg:p   eg:o .
```

3.
```
eg:s   eg:p   eg:s .
_:1    eg:p   [ eg:p   [] ] .
```

4.
```
eg:s   eg:p   eg:o .
_:1    eg:p   [ eg:p   [] ] .
```

Task 4: b

RDF Leanness II

Which of the following RDF graphs is not lean?

1.
```
eg:s   eg:p   eg:o .
_:1    eg:p   _:1  .
```

2.
```
eg:s   eg:p   _:2  .
_:1    eg:p   eg:o .
```

3.
```
eg:s   eg:p   eg:s .
_:1    eg:p   [ eg:p   [] ] .
```

4.
```
eg:s   eg:p   eg:o .
_:1    eg:p   [ eg:p   [] ] .
```

Solution

Graph 3 is not lean: the instance mapping all blank nodes to `eg:s` maps to the only triple `eg:s eg:p eg:s`, which is a proper subset of the graph.

Task 5: a

Cypher Querying

Consider a property graph that uses the `HAS_CHILD` relationship type to model parent–child relationships. Write a Cypher query that finds persons and their great-grandparents.

Task 5: a

Cypher Querying

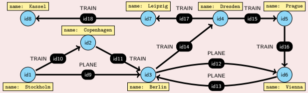
Consider a property graph that uses the HAS_CHILD relationship type to model parent–child relationships. Write a Cypher query that finds persons and their great-grandparents.

Solution

```
MATCH (person)<-[:HAS_CHILD*3]-(greatGrandparent)
RETURN person, greatGrandparent
```

Task 5: b

Cypher Query Evaluation

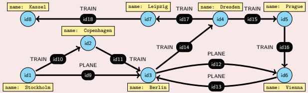


Which answers does the following Cypher query produce on this graph?

```
MATCH p = (s {name: "Stockholm"})-[:TRAIN*]-(a)-[:PLANE*1..]->(b)-[:TRAIN*0..]-(d {name: "Vienna"})
RETURN [ n IN nodes(p) | n.name ]
UNION ALL MATCH p = (s {name: "Stockholm"})-[:TRAIN*]-(d {name: "Vienna"})
RETURN [ n IN nodes(p) | n.name ]
```

Task 5: b

Cypher Query Evaluation



Which answers does the following Cypher query produce on this graph?

```
MATCH p = (s {name: "Stockholm"})—[:TRAIN∗]—(a)—[:PLANE∗1..]—>(b)—[:TRAIN∗0..]—(d {name: "Vienna"})
RETURN [ n IN nodes(p) | n.name ]
UNION ALL MATCH p = (s {name: "Stockholm"})—[:TRAIN∗]—(d {name: "Vienna"})
RETURN [ n IN nodes(p) | n.name ]
```

Solution

- ["Stockholm", "Copenhagen", "Berlin", "Vienna"]
- ["Stockholm", "Copenhagen", "Berlin", "Dresden", "Prague", "Vienna", "Berlin", "Vienna"]
- ["Stockholm", "Copenhagen", "Berlin", "Vienna", "Berlin", "Dresden", "Prague", "Vienna"]
- ["Stockholm", "Copenhagen", "Berlin", "Dresden", "Prague", "Vienna"]

Task 6: a

Datalog Querying

Consider a schema consisting of two unary predicates `first` and `last`, and of a binary predicate `next`. Let D be a database of that schema, encoding a linear order of the form $1 < 2 < 3 < \cdots < n-1 < n$ using facts

$$\texttt{first}(1) \qquad \texttt{next}(1,2) \qquad \texttt{next}(2,3) \qquad \cdots \qquad \texttt{next}(n-1,n) \qquad \texttt{last}(n)$$

Write a Datalog program P such that $\langle P, \texttt{Result}\rangle$ derives `Result` over D iff the linear order encoded by D has even length. If D is not of the form described above, the behaviour of P is unspecified.

Task 6: a

Datalog Querying

Consider a schema consisting of two unary predicates first and last, and of a binary predicate next. Let D be a database of that schema, encoding a linear order of the form $1 < 2 < 3 < \cdots < n-1 < n$ using facts

$$\texttt{first}(1) \qquad \texttt{next}(1,2) \qquad \texttt{next}(2,3) \qquad \cdots \qquad \texttt{next}(n-1,n) \qquad \texttt{last}(n)$$

Write a Datalog program P such that $\langle P, \texttt{Result}\rangle$ derives Result over D iff the linear order encoded by D has even length. If D is not of the form described above, the behaviour of P is unspecified.

Solution

```
odd(?X)  :- first (?X) .
odd(?Y)  :- next(?X, ?Y), even(?X) .
even(?Y) :- next(?X, ?Y), odd(?X) .
Result   :- even(?X), last (?X) .
```

Task 6: b

Stratified Datalog

Give a stratification of the following Datalog program P.

```
mother(?x, ?y)         :- triple(?x, wdt:P25, ?y) .
father(?x, ?y)         :- triple(?x, wdt:P22, ?y) .
notSameMother(?x, ?y) :- mother(?x, ?z), ~mother(?y, ?z) .
sameFather(?x, ?y)     :- father(?x, ?z), father(?y, ?z) .
notSameFather(?x, ?y) :- ~sameFather(?x, ?y) .
halfSiblings(?x, ?y)   :- sameMother(?x, ?y), notSameFather(?x, ?y) .
halfSiblings(?x, ?y)   :- sameFather(?x, ?y), notSameMother(?x, ?y) .
```

Task 6: b

Stratified Datalog

Give a stratification of the following Datalog program P.

```
mother(?x, ?y)        :- triple(?x, wdt:P25, ?y) .
father(?x, ?y)        :- triple(?x, wdt:P22, ?y) .
notSameMother(?x, ?y) :- mother(?x, ?z), ~mother(?y, ?z) .
sameFather(?x, ?y)    :- father(?x, ?z), father(?y, ?z) .
notSameFather(?x, ?y) :- ~sameFather(?x, ?y) .
halfSiblings(?x, ?y)  :- sameMother(?x, ?y), notSameFather(?x, ?y) .
halfSiblings(?x, ?y)  :- sameFather(?x, ?y), notSameMother(?x, ?y) .
```

Solution

Using three strata, a possible stratification maps predicates as follows:

1. `triple`, `mother`, `father`, `sameFather`
2. `notSameMother`, `notSameFather`
3. `sameMother`, `halfSiblings`

Task 7

Complexity of Problems

1. **given** a SPARQL query q, **decide** whether q has a match on the empty RDF graph
2. **given** a database instance $\mathcal{I}$, **decide** whether a fixed Datalog query $\langle P, \texttt{Result}\rangle$ derives $\texttt{Result}$ on $\mathcal{I}$
3. **given** a simple graph G, **decide** whether G has a 3-colouring
4. **given** a database instance $\mathcal{I}$ and a Datalog query $\langle P, \texttt{Result}\rangle$, **decide** whether $\langle P, \texttt{Result}\rangle$ derives $\texttt{Result}$ on $\mathcal{I}$

Sort the problems by their computational complexity, from the easiest to the hardest. That is, every problem should provably be at most as hard as all the problems following it.

Task 7

Complexity of Problems

1. **given** a SPARQL query q, **decide** whether q has a match on the empty RDF graph
2. **given** a database instance $\mathcal{I}$, **decide** whether a fixed Datalog query $\langle P, \texttt{Result}\rangle$ derives $\texttt{Result}$ on $\mathcal{I}$
3. **given** a simple graph G, **decide** whether G has a 3-colouring
4. **given** a database instance $\mathcal{I}$ and a Datalog query $\langle P, \texttt{Result}\rangle$, **decide** whether $\langle P, \texttt{Result}\rangle$ derives $\texttt{Result}$ on $\mathcal{I}$

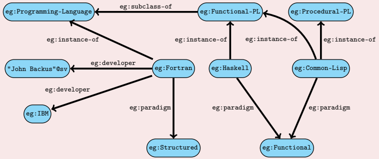
Sort the problems by their computational complexity, from the easiest to the hardest. That is, every problem should provably be at most as hard as all the problems following it.

Solution

1. **given** a database instance $\mathcal{I}$, **decide** whether a fixed Datalog query $\langle P, \texttt{Result}\rangle$ derives $\texttt{Result}$ on $\mathcal{I}$
2. **given** a simple graph G, **decide** whether G has a 3-colouring
3. **given** a SPARQL query q, **decide** whether q has a match on the empty RDF graph
4. **given** a database instance $\mathcal{I}$ and a Datalog query $\langle P, \texttt{Result}\rangle$, **decide** whether $\langle P, \texttt{Result}\rangle$ derives $\texttt{Result}$ on $\mathcal{I}$

Question 15

ShEx Evaluation

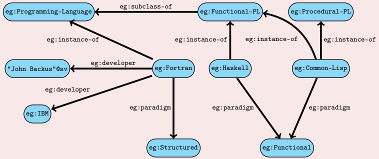



```
eg:programming-language {
  (eg:instance-of @<#programming_language> |
   eg:instance-of @<#subclass_of_programming_language>)+;
  eg:developer IRI*;
  eg:paradigm @<#programming_paradigm>*;
  eg:publication-date LITERAL*;
} <#subclass_of_programming_language> {
  (eg:subclass-of @<#programming_language> ; eg:subclass-of IRI *) |
  (eg:subclass-of @<#subclass_of_programming_language>; eg:subclass-of IRI *)
} <#programming_language> [ eg:Programming-Language ]
<#programming_paradigm> [ eg:Functional eg:Structured ]
```

Validate the RDF graph according to this schema, i.e., apply the `eg:programming-language` shape to the nodes `eg:Fortran`, `eg:Common-Lisp`, and `eg:Haskell`. Which of the nodes is valid and which is invalid for the schema? Explain your answer in each case.

Question 15

ShEx Evaluation




```
eg:programming−language {
   (eg:instance−of @<#programming_language> |
    eg:instance−of @<#subclass_of_programming_language>)+;
   eg:developer IRI∗;
   eg:paradigm @<#programming_paradigm>∗;
   eg:publication−date LITERAL∗;
} <#subclass_of_programming_language> {
   (eg:subclass−of @<#programming_language> ; eg:subclass−of IRI ∗) |
   (eg:subclass−of @<#subclass_of_programming_language>; eg:subclass−of IRI ∗)
} <#programming_language> [ eg:Programming−Language ]
<#programming_paradigm> [ eg:Functional eg:Structured ]
```

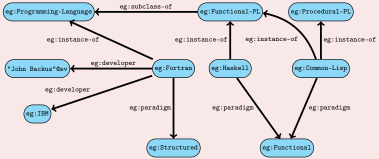
Validate the RDF graph according to this schema, i.e., apply the `eg:programming-language` shape to the nodes `eg:Fortran`, `eg:Common-Lisp`, and `eg:Haskell`. Which of the nodes is valid and which is invalid for the schema? Explain your answer in each case.

Solution

- `eg:Fortran` is invalid, `"John Backus"@sv` is not an IRI

Question 15

ShEx Evaluation




```
eg:programming−language {
  (eg:instance−of @<#programming_language> |
   eg:instance−of @<#subclass_of_programming_language>)+;
  eg:developer IRI∗;
  eg:paradigm @<#programming_paradigm>∗;
  eg:publication−date LITERAL∗;
} <#subclass_of_programming_language> {
  (eg:subclass−of @<#programming_language> ; eg:subclass−of IRI ∗) |
  (eg:subclass−of @<#subclass_of_programming_language>; eg:subclass−of IRI ∗)
} <#programming_language> [ eg:Programming−Language ]
<#programming_paradigm> [ eg:Functional eg:Structured ]
```

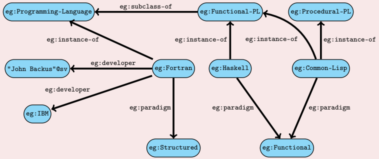
Validate the RDF graph according to this schema, i.e., apply the `eg:programming-language` shape to the nodes `eg:Fortran`, `eg:Common-Lisp`, and `eg:Haskell`. Which of the nodes is valid and which is invalid for the schema? Explain your answer in each case.

Solution

- `eg:Common-Lisp` is invalid, `eg:Procedural-Programming-Language` is not a `eg:subclass-of` of `eg:Programming-Language`

Question 15

ShEx Evaluation




```
eg:programming—language {
  (eg:instance—of @<#programming_language> |
   eg:instance—of @<#subclass_of_programming_language>)+;
  eg:developer IRI*;
  eg:paradigm @<#programming_paradigm>*;
  eg:publication—date LITERAL*;
} <#subclass_of_programming_language> {
  (eg:subclass—of @<#programming_language> ; eg:subclass—of IRI *) |
  (eg:subclass—of @<#subclass_of_programming_language>; eg:subclass—of IRI *)
} <#programming_language> [ eg:Programming—Language ]
<#programming_paradigm> [ eg:Functional eg:Structured ]
```

Validate the RDF graph according to this schema, i.e., apply the `eg:programming-language` shape to the nodes `eg:Fortran`, `eg:Common-Lisp`, and `eg:Haskell`. Which of the nodes is valid and which is invalid for the schema? Explain your answer in each case.

Solution

- `eg:Haskell` is valid: both `eg:paradigm` and `eg:instance-of` comply with the schema