Minimal type inference for Linked Data consumers

and

A descriptive type foundation for RDF Schema

Articles published in Journal of Logical and Algebraic Methods in Programming

Gabriel Ciobanu\(^{(a)}\)  Ross Horne\(^{(a)(b)(c)}\)  Vladimiro Sassone\(^{(d)}\)

\(^{(a)}\) Romanian Academy, Iaşi, Romania
\(^{(b)}\) Nanyang Technological University, Singapore
\(^{(c)}\) Kazakh British Technical University, Almaty, Kazakhstan
\(^{(d)}\) Electronics and Computer Science, University of Southampton, United Kingdom

26 February 2016

\(^{1}\) Corresponding author. Email: rhorne@ntu.edu.sg
History of the Web of Linked Data

1989–
The Web of Hypertext
Emphasis on documents interlinked using URIs.
Berners-Lee. Information management: A proposal

2001–2006
The Semantic Web
Emphasis on deep ontologies classifying everything (we can learn from an AI winter).

2006–
The Web of Linked Data
Emphasis on raw data interlinked using URIs and delivered by simple data APIs.
Berners-Lee. Linked Data — design issues
Four Principles of Linked Data

▶ Use URIs to identify resources.
▶ Use HTTP URIs to identify resources so we can look them up.
▶ When a URI is looked up, return data about the resource using the standards.
▶ Include URIs in the data, so they can also be looked up.
Dereferencing the URI  res:Kazakhstan


Request:

GET /resource/Kazakhstan HTTP/1.1
Host: dbpedia.org
Accept: text/n3

Response:

HTTP/1.1 303 See Other
Content-Type: text/n3
Location: http://dbpedia.org/data/Kazakhstan.n3
Dereferencing the URI \textit{res:Kazakhstan}

```shell
curl -H "Accept:text/n3" http://dbpedia.org/data/Kazakhstan.n3
```

```
@prefix dbpprop: <http://dbpedia.org/property/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
dbpedia:Medeo dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Zhetyсу_Стадион dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Aстана_Арена dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Казахстан_Спорт_Палас dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Мунайшы_Стадион dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Арал_Море dbpedia-owl:location dbpedia:Kazakhstan .
dbpedia:Арал_Море dbpedia-owl:country dbpedia:Kazakhstan .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ns4: <http://en.wikipedia.org/wiki/> .
ns4:Казахстан foaf:primaryTopic dbpedia:Kazakhstan .
dbpedia:Air_Kokshetau dbpprop:headquarters dbpedia:Kazakhstan .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ns6: <http://data.nytimes.com/> .
ns6:N63032621026086062091 owl:sameAs dbpedia:Kazakhstan .
dbpedia:Рахимжан_Кожоqарбаев dbpprop:placeOfBirth dbpedia:Kazakhstan .
@prefix yago-res: <http://mpii.de/yago/resource/> .
yago-res:Казахстан owl:sameAs dbpedia:Kazakhstan .
dbpedia:Регина_Куликова dbpedia-owl:birthPlace dbpedia:Kazakhstan .
dbpedia:Алматы_Международная_Школа dbpprop:country dbpedia:Kazakhstan .
dbpedia:Дарем_Дарел dbpedia-owl:country dbpedia:Kazakhstan .
dbpedia:Dmytro_Salamatin dbpedia-owl:birthPlace dbpedia:Kazakhstan .
dbpedia:Дунганский_язык dbpedia-owl:spokenIn dbpedia:Kazakhstan .
dbpedia:Казахстан dbpedia-owl:percentageOfAreaWater "1.7"~xsd:float .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix yago: <http://dbpedia.org/class/yago/> .
@prefix ns10: <http://umbel.org/umbel/rc/> .
dbpedia:Казахстан rdf:type ns10:Location_Underspecified .
dbpedia:Казахстан rdf:type dbpedia-owl:PopulatedPlace ,
dbpedia:Казахстан rdf:type yago:CentralAsianCountries ,
dbpedia:Казахстан rdf:type yago:LandlockedCountries .
@prefix ns11: <http://schema.org/> .
dbpedia:Казахстан rdf:type ns11:Country ,
dbpedia:Казахстан rdf:type dbpedia-owl:Country ,
dbpedia:Казахстан rdf:type yago:YagoGeoEntity ,
dbpedia:Казахстан rdf:type dbpedia-owl:Place ,
dbpedia:Казахстан rdf:type yago:Economy108366753 .
```
An Architecture for Linked Data Consumers

Front end: traditional Web architecture, with SPARQL replacing SQL.

Triples store: graph based query and update, suited to combining data sources so they can be collectively queried.

Back end: pulls raw data from the Web, using RESTful open data APIs.

Our problem: Make programming the back end easier.
Comparing SPARQL queries to background scripts.

A *front end* SPARQL query that discovers a URI for the capital of Kazakhstan:

```
select $x
from named res:Kazakhstan
where
  { graph res:Kazakhstan { res:Kazakhstan dbp:capital $x } }
limit 1
```

A *background* script that finds the URI for the capital of Kazakhstan, then loads dereferenced data into the triple store in a *named graph* identified by the discovered URI.

```
select $x
from named res:Kazakhstan
where
  graph res:Kazakhstan { res:Kazakhstan dbp:capital $x }
from named $x
```
Traditional type systems are Prescriptive.

```sql
do select $x: xsd:anyURI, $y: xsd:string
where
  $x rdfs:label $y
  langMatches($y, en)
  regex($y, ^cyber)
from $x
```

Types `xsd:anyURI` and `xsd:string` are prescriptive. The language and regular expressions only apply to strings, and only URIs can be dereferenced.

**Type checking:** If programmer wrongly writes

```sql
from $y
```

a type error is thrown since a string cannot be dereferenced.

Given the range of `rdfs:label` is `xsd:string`, if the programmer wrongly writes

```sql
$y rdfs:label $x
```

the type checker will detect the error.

**Type inference:** Given no type information, a type inference algorithm infers:

- $y$ must be of type `xsd:string` since used in a regular expression,
- $x$ must be of type `xsd:anyURI` since it is dereferenced,
- furthermore, the range of `rdfs:label` must be `xsd:string`. 
Programmers make more mistakes for larger scripts

```sql
from named res:Almaty
select $almalat: xsd:decimal, $almalong: xsd:decimal
where
  graph res:Almaty {res:Almaty geo:lat $almalat}
  graph res:Almaty {res:Almaty geo:long $almalong}
from named res:Kazakhstan
do select $loc: xsd:anyURI
where
  graph res:Kazakhstan {$loc dbp:location res:Kazakhstan}
from named $loc
select $lat: xsd:decimal, $long: xsd:decimal
where
  graph $loc {$loc geo:lat $lat}
  graph $loc {$loc geo:long $long}
  haversine($lat, $long, $almalat, $almalong) < 1000
do select $person: xsd:anyURI
where
  graph $loc {$person dbp:birthPlace $loc}
from named $person
```

Dereference data about people born in places in Kazakhstan within 1000km from Almaty.
Descriptive Types for RDF Schema: simple entailment

Data:

\[ \text{res:Vitali_Klitschko} \text{ dbp:boxerCategory} \text{ res:Heavyweight} \]

RDF Schema:

\[ \text{dbp:boxerCategory rdfs:domain dbp:Boxer} \]
\[ \text{dbp:boxerCategory rdfs:range dbp:BoxingCategory} \]

Rule:

\[ \text{uri}_1 \text{ rdfs:domain type} \text{ uri}_0 \text{ uri}_1 \text{ uri}_2 \]
\[ \text{uri}_0 \text{ a type} \]

Simply entailed type:

\[ \text{res:Vitali_Klitschko a dbp:Boxer} \]
Descriptive Types for RDF Schema: simple entailment

More data:

res:Vitali_Klitschko dbp:boxerCategory res:Heavyweight
res:Vitali_Klitschko dbp:birthPlace res:Kyrgyz_SSR

More RDF Schema:

dbp:boxerCategory rdfs:domain dbp:Boxer
dbp:boxerCategory rdfs:range dbp:BoxingCategory

res:Vitali_Klitschko a dbp:Boxer

dbp:birthPlace rdfs:domain dbp:Person
dbp:birthPlace rdfs:range dbp:Place

Simply entailed types:

res:Vitali_Klitschko a dbp:Boxer
res:Vitali_Klitschko a dbp:Person

or equivalently

res:Vitali_Klitschko a owl:intersectionOf( dbp:Boxer, dbp:Person )
More data:

res:Vitali_Klitschko dbp:boxerCategory res:Heavyweight
res:Vitali_Klitschko dbp:birthPlace res:Kyrgyz_SSR

More RDF Schema:

dbp:boxerCategory rdfs:domain dbp:Boxer
dbp:boxerCategory rdfs:range dbp:BoxingCategory

res:Vitali_Klitschko a dbp:Boxer

dbp:birthPlace rdfs:domain dbp:Person
dbp:birthPlace rdfs:range dbp:Place

Inferred subclass assumption:

dbp:Boxer rdfs:subClassOf dbp:Person
Descriptive Types for RDF Schema: well-typed data

Data about Vitali’s brother Wladimir:

res:Wladimir_Klitschko dbp:birthPlace dbp:Kazakh_SSR

RDF Schema information:

res:Wladimir_Klitschko a dbp:Boxer
dbp:Kazakh_SSR a dbp:Place
dbp:birthPlace rdfs:domain dbp:Person
dbp:birthPlace rdfs:range dbp:Place
dbp:Boxer rdfs:subClassOf dbp:Person

The following simply entailed type is redundant.

res:Wladimir_Klitschko a dbp:Person

We already know this from:

dbp:Boxer rdfs:subClassOf dbp:Person
res:Wladimir_Klitschko a dbp:Boxer

Typing: for well-typed systems, no new inferences need be applied.
More data about Vitali:

\[ \text{res:Vitali.Klitschko} \text{ free:government/politician/party free:m/0j1b9hc} \]

More RDF Schema information:

\[ \text{free:government/politician/party rdfs:domain free:government/politician} \]
\[ \text{free:government/politician/party rdfs:range free:government/political_party} \]

Options presented to the expert:

1. Use simple entailment:

\[ \text{res:Vitali.Klitschko a owl:intersectionOf( dbp:Boxer, free:government/politician) } \]

2. Use inferred subclass assumption:

\[ \text{dbp:Boxer rdfs:subClassOf free:government/politician} \]

3. Ignore warning and resolve later.

**Subjectivity:** Human experts know not every boxer is a politician. Machines don’t.
Example of Scripting Problem: want Andrei Ershov the scientist

- Initial script:

```python
from res:Andrei_Yershov
```

- Initial data including:

```plaintext
free:book.author.works_written rdfs:range free:book
```

Warning with options:
2. Relax domain of free:book.author.works_written to xsd:anyURI.
3. Ignore warning and resolve later.
Example of Scripting Problem: got Andrei Ershov the sportsman

- Second state of script:

  ```
  ```

- Data including triples from dereferencing res:Andrei_Yershov:

  ```
  res:Andrei_Yershov a dbp:IceHockeyPlayer
  free:book.author.works_written rdfs:range free:book
  ```

Warning with options (None make sense, suggesting a mistake!):

1. Refine type of res:Andrei_Yershov:

   ```
   res:Andrei_Yershov a owl:intersectionOf(
       yago:IceHockeyPlayer,
       free:book.author
   )
   ```

2. Relax domain of free:book.author.works_written:

   ```
   free:book.author.works_written rdfs:domain owl:unionOf(
       free:book.author,
       dbp:IceHockeyPlayer
   )
   ```

3. Refine subtype assumptions such that:

   ```
   dbp:IceHockeyPlayer rdfs:subClassOf free:book.author
   ```
Conclusion: Prescriptive and Descriptive Types

**Prescriptive types:**
- Traditional type system aware of simple datatypes, such as integers and strings.
- Helps avoid writing scripts that can never work.
- Type inference makes programmer’s life easier, by automatically calculating types.

**Descriptive types:**
- Novel aspects of RDF Schema types, concerning URIs.
- Helps make sense of subjective knowledge consumed from the Web.
- Throws warnings with menu of options rather than errors.
- RDF Schema simple entailment is one of several inference modes.
- Accommodates more data publishing models than W3C standards.
- Philosophically, narrows gap between type theory and semiotics.

**Implementation:**
- Mature language specification ready to be implemented.
- Techniques can be applied to extensions of popular scripting languages.