

Towards a Semantic Web Toolkit in Haskell

Querying RDF graphs & SPARQL construction

Rob Stewart

Heriot Watt University

Updated: 29th September 2012

Preamble

- Who I am...
 - ▶ Work @ Heriot Watt University
 - ★ **PhD** Fault tolerant, distributed-memory Haskell computing
 - ★ **Researcher** SerenA project (*more on that later*).
 - ▶ I'm on Twitter: @robstewartUK
 - ▶ I'm also on Google+: <http://goo.gl/eL8zg>
- What I'll talk about
 - ▶ The *semantic web*
 - ▶ Linked Open Data
 - ▶ Available Haskell libraries to poke at both
- LaTeX, Haskell & Java source code on GitHub:
<https://github.com/robstewart57/edlambda-semanticweb-haskell>

The Semantic Web

The Semantic Web

What is it?



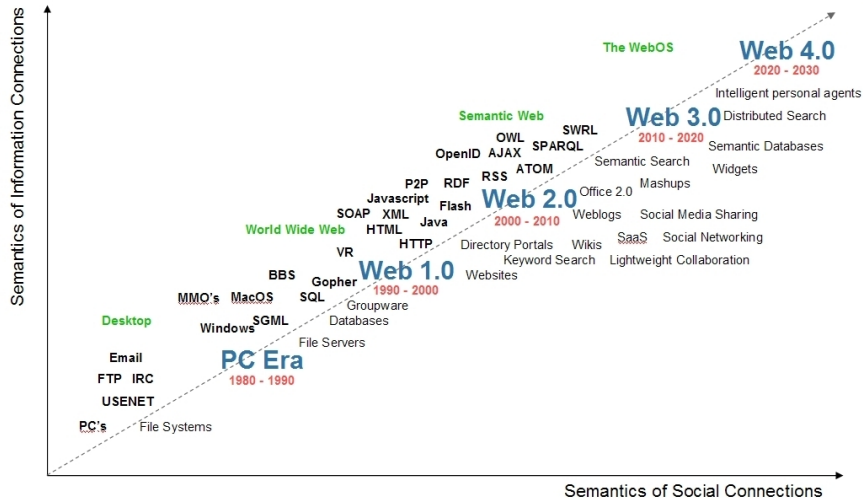
- A collaborative movement led by W3C
- Promotes common data formats
- Moving from *unstructured documents* into a **web of data**.
- “Web 3.0” is sometimes used as a synonym for the semantic web.

*The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries*¹.

¹“W3C Semantic Web Activity”. World Wide Web Consortium (W3C). November 7, 2011

Future of the Web

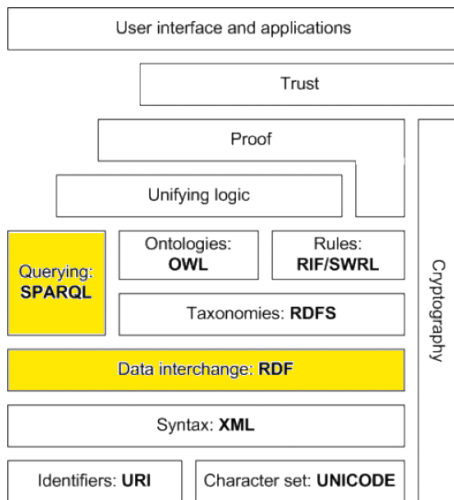
Web connectivity



Source: Radar Networks & Nova Spivack, 2007 – www.radarnetworks.com

The Semantic Web

Semantic Web Stack



The Semantic Web

Principles

- **URI** is simply a Web identifier
 - ▶ A disambiguation mechanism for distributed data
 - ▶ Start with *http://* or *ftp://* ...
 - ▶ Anyone can create a URI! e.g.
 - ★ `https://twitter.com/#!/robstewartUK`
 - ★ `http://dblp.l3s.de/d2r/page/authors/Tim_Berners-Lee`

²though not yet standardised

The Semantic Web

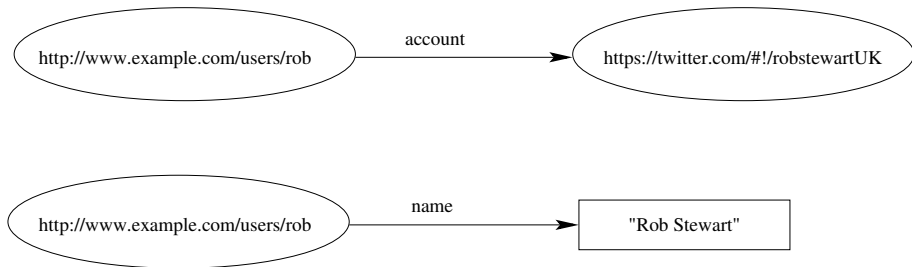
Principles

- **URI** is simply a Web identifier
 - ▶ A disambiguation mechanism for distributed data
 - ▶ Start with *http://* or *ftp://* ...
 - ▶ Anyone can create a URI! e.g.
 - ★ `https://twitter.com/#!/robstewartUK`
 - ★ `http://dblp.l3s.de/d2r/page/authors/Tim_Berners-Lee`
- **RDF** statements consist of three parts:
 - Subject** is the *resource* being described
 - Predicate** indicates a relationship with...
 - Object** is either a resource or a *literal*
- ▶ RDF can be serialised in XML, N3, Turtle and JSON².

²though not yet standardised

The Semantic Web

Two RDF triples



The Semantic Web

Resource Description Framework serialised in XML

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:RDF>
    <rdf:Description about="http://www.example.com/users/rob">
      <foaf:name>Rob Stewart</foaf:name>
      <foaf:account
        rdf:resource="https://twitter.com/#!/robstewartUK" />
    </rdf:Description>
  </rdf:RDF>
```

The Semantic Web

Resource Description Framework serialised in XML

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:RDF>
    <rdf:Description about="http://www.example.com/users/rob">
      <foaf:name> Rob Stewart </foaf:name>
      <foaf:account
        rdf:resource="https://twitter.com/#!/robstewartUK" />
    </rdf:Description>
  </rdf:RDF>
```

Key

Prefixes

Subject

Predicate

Object

SPARQL Protocol and RDF Query Language

- And RDF query language to retrieve and manipulate persisted RDF
- On 15th January 2008, SPARQL 1.0 became W3C recommendation
- Four query forms
 - ▶ SELECT Extracts raw values to a table format
 - ▶ CONSTRUCT Extracts raw values into RDF
 - ▶ ASK Queries for the existence of a triple
 - ▶ DESCRIBE Extracts triples where specified URI is subject

The Semantic Web

SPARQL SELECT

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?account
WHERE {
  ?person foaf:name ?name
  ?person foaf:account ?account
}
```

name	account
"Rob Stewart"	<https://twitter.com/#!/robstewartUK>

The Semantic Web

SPARQL DESCRIBE

DESCRIBE <http://www.example.com/users/rob>

```
<?xml version="1.0"?>
```

```
<rdf:RDF
```

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
```

```
  <rdf:RDF>
```

```
    <rdf:Description about="http://www.example.com/users/rob">
```

```
      <foaf:name>Rob Stewart</foaf:name>
```

```
      <foaf:account
```

```
        rdf:resource="https://twitter.com/#!/robstewartUK" />
```

```
    </rdf:Description>
```

```
</rdf:RDF>
```

The Semantic Web

SPARQL CONSTRUCT

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX example: <http://www.example.org/onto/>
CONSTRUCT { ?person example:hasName ?name }
WHERE {
    ?person foaf:name ?name
}
```

name	account
"Rob Stewart"	<https://twitter.com/#!/robstewartUK>

The Semantic Web

SPARQL ASK

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
ASK { ?x foaf:name "Rob Stewart" }
```

Yes.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
ASK { ?x foaf:topic_interest  
      <http://www.dbpedia.org/resource/Closed_source_software> }
```

No.

Linked Open Data

Linked Open Data

What is it?

- Linked Data
 - ▶ Semantic structured data...
 - ▶ Using semantic web principles (e.g. RDF)
 - ▶ Referencing disambiguated URIs means data is interlinked

Linked Open Data

What is it?

- Linked Data

- ▶ Semantic structured data...
- ▶ Using semantic web principles (e.g. RDF)
- ▶ Referencing disambiguated URIs means data is interlinked

- Open Data

- ▶ Certain data should be freely available to use and republish
- ▶ Without copyright, patents, or imposed control
- ▶ Philosophy correlates to *open source code & open access*.

Linked Open Data

What is it?

- Linked Data

- ▶ Semantic structured data...
- ▶ Using semantic web principles (e.g. RDF)
- ▶ Referencing disambiguated URIs means data is interlinked

- Open Data

- ▶ Certain data should be freely available to use and republish
- ▶ Without copyright, patents, or imposed control
- ▶ Philosophy correlates to *open source code & open access*.

- Linked Open Data!

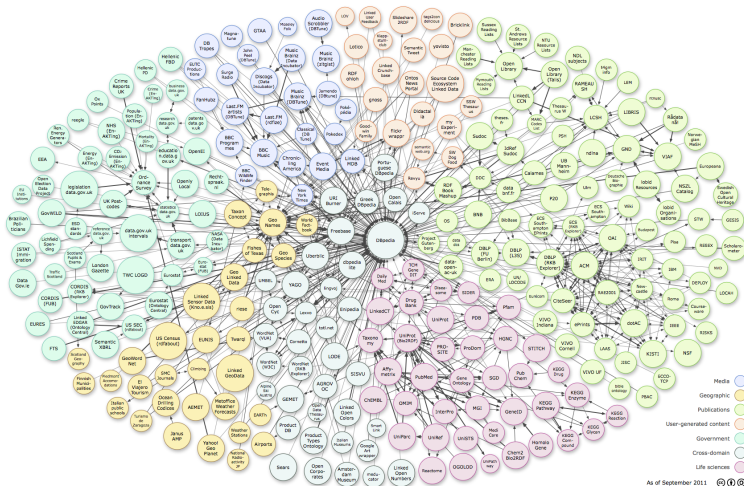
- ▶ LOD project aims to publish open data sets as RDF. Data sources include:
 - ★ DBpedia
 - ★ Geonames
 - ★ MusicBrainz
 - ★ DBLP

In September 2007



Linked Open Data

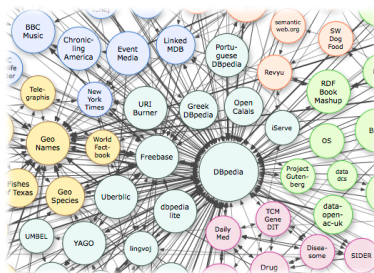
In September 2011



As of September 2011 © ⓘ ⓘ

Linked Open Data

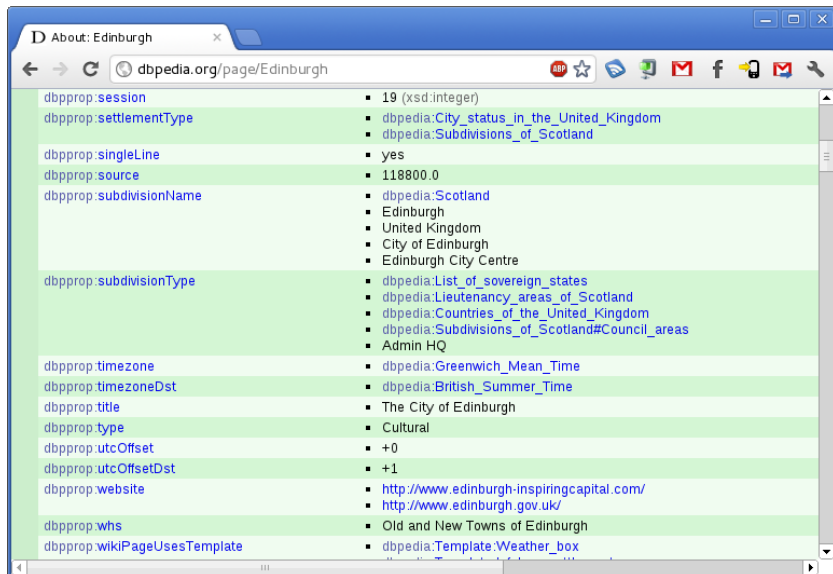
What is DBpedia?



- Extracts structured content from *Wikipedia* pages.
- Query relationships associated with Wikipedia resources. . .
- Including links to other semantic datasets (which is cool!).
- Describes *3.64 million* things, including. . .
 - ▶ 416,000 persons
 - ▶ 526,000 places
 - ▶ and 169,000 organisations

Linked Open Data

Resource Example - <http://dbpedia.org/resource/Edinburgh>



The screenshot shows a web browser window with the address bar displaying dbpedia.org/page/Edinburgh. The page content is a structured list of properties and their values, organized into rows with alternating light green and white backgrounds. The properties are listed on the left, and their corresponding values are listed on the right, often as bulleted lists.

dbpprop:session	19 (xsd:integer)
dbpprop:settlementType	<ul style="list-style-type: none">dbpedia:City_status_in_the_United_Kingdomdbpedia:Subdivisions_of_Scotland
dbpprop:singleLine	yes
dbpprop:source	118800.0
dbpprop:subdivisionName	<ul style="list-style-type: none">dbpedia:ScotlandEdinburghUnited KingdomCity of EdinburghEdinburgh City Centre
dbpprop:subdivisionType	<ul style="list-style-type: none">dbpedia:List_of_sovereign_statesdbpedia:Lieutenancy_areas_of_Scotlanddbpedia:Countries_of_the_United_Kingdomdbpedia:Subdivisions_of_Scotland#Council_areasAdmin HQ
dbpprop:timezone	dbpedia:Greenwich_Mean_Time
dbpprop:timezoneDst	dbpedia:British_Summer_Time
dbpprop:title	The City of Edinburgh
dbpprop:type	Cultural
dbpprop:utcOffset	+0
dbpprop:utcOffsetDst	+1
dbpprop:website	<ul style="list-style-type: none">http://www.edinburgh-inspiringcapital.com/http://www.edinburgh.gov.uk/
dbpprop:whs	Old and New Towns of Edinburgh
dbpprop:wikiPageUsesTemplate	dbpedia:Template:Weather_box

Linked Open Data

Linked Geo Data Project



- Effort to add spatial dimension to *web of data*
- Uses *crowd sourced* information from OpenStreetMap
- Interlinked resources with other knowledge bases on LOD network
- Information of 350 million nodes and 30 million ways. . .
- Resulting in over 2 billion triples!

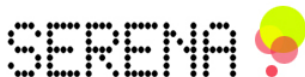
Linked Open Data

Linked Geo Data Project

The screenshot displays the LinkedGeoData Browser interface. The search bar at the top contains the text "edinburgh". Below the search bar, a grid of nine location cards is shown, each with a red pin icon and a label: "Edinburgh City of Edinburgh", "Edinburgh Johnson County", "Edinburgh Hoke", "Edinburgh Richland", "Edinburgh Wayne", "Edinburgh Johnson", "Edinburgh 5111", "Edinburgh Toledo", and "Edinburgh Green". To the left of the map, a list of 34 instances is displayed, including "New Town", "Edinburgh Central Youth", "Urban Angel", "Cumberland Bar", "Nelson Monument", "National Monument", "the dogs", "D.J. Alexander", "Bourne", "Ryden Lettings", "Anthony Wood", "Glass & Thompson", "Twenty Twenty", "Amore Dogs", "Food Glorious Food", "Shilla", "Kweilin Cantonese Resta", "Veterinary Surgery", "Christopher Ness Jew", "The Wally Dug", "Clydesdale Bank", "Bellevue Medical Centre", "Vittoria", "Black Dutch cafe", "Portuguese Cannon", "Playfair Monument", "Valvona & Crolla", "Greenside Parish Church", "Calton Hill", "Wilson's Corner Store", "Hertz", "Coffee Republic", "Melville's Monument", and "Anwar-E-Madina". On the right side, a "Facets" panel lists various categories and their counts: "Node (137)", "Place (2)", "Amenity (118)", "Tourism (10)", "ManMade (1)", "Historic (4)", "Historicannon (1)", "Natural (1)", and "Landuse (1)". The main map area shows a detailed view of Edinburgh, with red pins indicating the locations of the data points. The map includes street names, landmarks like Calton Hill, and a scale bar. The bottom right corner of the map displays the coordinates "-3.17835, 55.95552" and the "AKSW" logo.

Linked Open Data - Project case study

SerenA - *Chance Encounters in the Space of Ideas*



- Homepage: <http://www.serena.ac.uk>
- Aims to bring closer people with *ideas, places, relevant documents, people*. . . resources!
- Presentation of ideas and resources, plus explanation of chain reasoning is *really* important for perceived **serendipity**.
- Implementation efforts are in progress. . .
 - ▶ Information seeking on cloud of *Linked Open Data*
 - ▶ *Agent framework* used for autonomous search and interface scheduling

For more implementation info: <http://goo.gl/ZGsfJ>

Semantic Web

Real World Applications

• Web search

- ▶ `schema.org` used by Microsoft, Google and Yahoo.

“A shared markup vocabulary makes easier for webmasters to decide on a markup schema and get the maximum benefit for their efforts”³.

• Retail

- ▶ `http://www.bestbuy.com`

- ★ Exposes RDF describing their products
- ★ 450,000 items, 60 triples per item... 27 million triples!

“Its easy to sell the top 20 products using keywords, but surfacing the long tail is where the data makes the difference.” - Jay Myers, Best Buy⁴

³<http://schema.org/>

⁴Best Buy Adopts RDF and the Open Web <http://goo.gl/EWar>

Semantic Web

JISC Monitoring Unit Data as RDF

- Project leader: Wilbert Kraan (@wilm on Twitter)
- JISCMU...

supports JISC policy and planning by providing a quality assurance role in monitoring the performance and use of JISC services. ⁵

- Example - full *institute* data as RDF/XML
 - ▶ <http://data.jiscmu.ac.uk/rest/organisations/>

```
SELECT ?Institution ?p ?o
WHERE{
?Institution a mu:Organisation .
?Institution mu:OfficialName "University of Edinburgh" .
?Institution ?p ?o . }
```

⁵<http://www.jiscmu.ac.uk/about/>

Semantic Web

Linked Data: JISC Institute data

Institution	p	o
http://data.jiscmu.ac.uk/rest/organisations/id/83	http://purl.org/dc/dcam/memberOf	SUNCAT Contributing Library
http://data.jiscmu.ac.uk/rest/organisations/id/83	http://purl.org/dc/dcam/memberOf	universitas 21
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:AlphabetisedName	Edinburgh, University of
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Country	Scotland
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Fax	0131 650 2147
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:JiscBanding	A
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:JiscFundedJanetConnection	true
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:LastChange	Join Organisation To
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:MailPoint	Old College
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:NetworkRegion	EaStMAN
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:OfficialName	University of Edinburgh
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:PostCode	EH8 9YL
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Sector	Higher Education
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Street	South Bridge
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Telephone	0131 650 1000
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Territory	Edinburgh & Lothian
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Town	Edinburgh
http://data.jiscmu.ac.uk/rest/organisations/id/83	mu:Website	http://www.ed.ac.uk/

Semantic Web Toolkits

Semantic Web Toolkits

Popular Toolkits

Functional languages

- scardf (Scala)
- Wilbur (Common Lisp)
- Clojure-rdf (Clojure)
- *(Nothing for Erlang?)*

(Other languages)

- Jena (Java)
- Sesame (Java)
- Redland (C)
- rdfquery (Javascript)
- RAP (php)
- SWI-Prolog (prolog)
- Many more...

Jena

A very popular semantic web framework in Java

“ Semantic Web applications. Jena provides a collection of tools and Java libraries to help you to develop semantic web and linked-data apps, tools and servers.”

- Support for serialisation of RDF in XML, N-triples and Turtle formats
- Ontology API for handling OWL and RDFS ontologies
- Rule-based inference engine for reasoning with RDF and OWL
- Query engine compliant with the latest SPARQL specification
- Lots more...

We'll be comparing RDF4H with Jena for RDF querying and manipulation.

Semantic Web Toolkits

Haskell packages

`rdf4h` is a library for working with RDF in Haskell.

`hsparql` includes a DSL to easily create queries, as well as methods to submit those queries to a SPARQL server, returning the results as simple Haskell data structures

`swish` is another library for working with RDF in Haskell. It explores Haskell as "a scripting language for the Semantic Web".

Haskell libraries

Looking at rdf4h

- **Author** Calvin Smith
- **Maintainer** Rob Stewart
- **Source code**

- ▶ Rob (upstream)

- ★ `git clone git://github.com/robstewart57/rdf4h.git`

Semantic Web Toolkits

RDF4H

rdf4h is a library for working with RDF in Haskell.

- Supports parsing and serialisation of file formats:
 - ▶ N-Triples
 - ▶ Turtle
 - ▶ XML
- Provides type representation of:
 - ▶ RDF triples
 - ▶ Prefix mappings; namespaces ...
- Querying over data:
 - ▶ Sorting triples
 - ▶ Finding resources
 - ▶ Isomorphic graph equality
 - ▶ And other stuff...

RDF4H

Types

```
data Triple =  
    Triple !Node !Node !Node  
  
type Triples = [Triple]  
type Subject = Node  
type Predicate = Node  
type Object = Node  
  
data Node =  
  
    -- |An RDF URI reference. See  
    -- <http://www.w3.org/TR/rdf-concepts/#section-Graph-URIref> for more  
    -- information.  
    UNode !T.Text  
  
    -- |An RDF blank node. See  
    -- <http://www.w3.org/TR/rdf-concepts/#section-blank-nodes> for more  
    -- information.  
    | BNode !T.Text  
  
    -- |An RDF blank node with an auto-generated identifier, as used in  
    -- Turtle.  
    | BNodeGen !Int
```

RDF4H

RDF type class

```
class RDF rdf where
  -- |Return the base URL of this graph, if any.
  baseUrl :: rdf → Maybe BaseUrl

  -- |Return the prefix mappings defined for this graph, if any.
  prefixMappings :: rdf → PrefixMappings

  -- |Merges specified prefix mappings with the existing mappings
  addPrefixMappings :: rdf → PrefixMappings → Bool → rdf

  -- |Return an empty RDF graph.
  empty :: rdf

  -- |Return a graph containing all the given triples.
  mkRdf :: Triples → Maybe BaseUrl → PrefixMappings → rdf

  -- |Return all triples in the graph, as a list.
  triplesOf :: rdf → Triples

  -- |Return the triples in the graph that match the given pattern.
  query :: rdf → Maybe Subject → Maybe Predicate → Maybe Object → Triples
```

RDF4H

RDF instances

TriplesGraph contains a list-backed graph implementation suitable for smallish graphs or for temporary graphs that will not be queried. If you might have duplicate triples, use **MGraph** instead, which is also more efficient.

MGraph A simple graph implementation backed by `Data.Map`.

RDF4H

Useful functions

```
-- |Return a URIRef node for the given bytestring URI.
unode :: T.Text → Node

-- |Return a blank node using the given string identifier.
bnode :: T.Text → Node

-- |Return a literal node using the given LValue.
lnode :: LValue → Node

-- |Constructor functions for LValue
plainL :: T.Text → LValue
plainLL :: T.Text → T.Text → LValue
typedL :: T.Text → T.Text → LValue

-- |A constructor function for 'Triple'
triple :: Subject → Predicate → Object → Triple

subjectOf :: Triple → Node
predicateOf :: Triple → Node
objectOf :: Triple → Node

isIsomorphic :: forall rdf1 rdf2. (RDF rdf1, RDF rdf2) ⇒ rdf1 → rdf2 → Bool
```


Comparing RDF4H and Jena

Reading an RDF file

Java

```
final public Model readFile(String filename) throws FileNotFoundException {  
    return ModelFactory.createDefaultModel().read(new FileReader("file.nt"), null);  
}
```

Haskell

```
readRDFFile :: IO TriplesGraph  
readRDFFile = fmap fromEither (parseFile NTriplesParser "file.nt")
```

Comparing RDF4H and Jena

Extracting Triples from Graphs

Java

```
final public Collection<Triple> getTriples(Model model){  
    Collection<Triple> triples = new LinkedList<Triple>();  
    StmtIterator stmts = model.listStatements();  
    Statement s;  
    while (stmts.hasNext()){  
        s = stmts.nextStatement();  
        triples.add(s.asTriple());  
    }  
    return triples;  
}
```

Haskell

```
getTriples :: TriplesGraph → Triples  
getTriples graph = triplesOf graph
```

Comparing RDF4H and Jena

Checking node type

Java

```
final public String checkNodeType(Node node){  
    String str = null;  
    if (node.isURI())      str = node + " is URI";  
    else if (node.isBlank()) str = node + " is blank";  
    else if (node.isLiteral()) str = node + " is literal";  
    return str;  
}
```

Haskell

```
checkNodeType :: Node → String
```

```
checkNodeType node
```

```
| isUNode node = (show node) ++ " is URI"  
| isBNode node = (show node) ++ " is blank"  
| isLNode node = (show node) ++ " is literal"
```

Comparing RDF4H and Jena

Constructing triples with language literal

Java

```
final public Triple makeTriple(String s, String p, String oLangLiteral){  
    Model model = ModelFactory.createDefaultModel();  
    Node subj = model.createResource(s).asNode();  
    Node pred = model.createResource(p).asNode();  
    Node obj = model.createLiteral(oLangLiteral,"en").asNode();  
    return new Triple(subj,pred,obj);  
}
```

Haskell

```
mkTriple :: String → String → String → Triple  
mkTriple s p o =  
    let subj = unode (s2b s)  
        pred = unode (s2b p)  
        obj = lnode (plainLL (s2b o) (s2b "en"))  
    in triple subj pred obj
```

Comparing RDF4H and Jena

Constructing RDF graphs in Java

```
final public Model mkModel(){
    Model model = ModelFactory.createDefaultModel();
    String foafNS = "http://xmlns.com/foaf/0.1/";
    model.setNsPrefix("foaf", foafNS);
    Property topic_interest = model.createProperty(foafNS, "topic_interest");
    Property account = model.createProperty(foafNS, "account");

    Resource me = model.createResource("http://example.org/users/robstewart");
    Resource twitter = model.createResource("https://twitter.com/#!/robstewartUK");
    Resource haskell = model.createResource(
        "http://dbpedia.org/resource/Haskell_(programming_language)");

    Statement stmt;
    stmt = model.createStatement(me, account, twitter);
    model.add(stmt);
    stmt = model.createStatement(me, topic_interest, haskell);
    model.add(stmt);
    return model;
}
```

Comparing RDF4H and Jena

Constructing RDF graphs in Haskell

```
mkGraph :: TriplesGraph
mkGraph = mkRdf
  [Triple
    ((unode o s2b) "http://example.org/users/robstewart")
    ((unode o s2b) "foaf:topic_interest")
    ((unode o s2b) "http://dbpedia.org/resource/Haskell_(programming_language)"),
    Triple
    ((unode o s2b) "http://example.org/users/robstewart")
    ((unode o s2b) "foaf:account")
    ((unode o s2b) "https://twitter.com/#!/robstewartUK")
  ]
Nothing
(PrefixMappings
  (Map.fromList [ (s2b "foaf", s2b "http://xmlns.com/foaf/0.1/") ]))
```

Haskell libraries

Looking at swish

- **Author** Graham Klyne
- **Maintainers** Doug Burke
- **Source code**

- ▶ Doug

- ★ `hg clone https://bitbucket.org/doug_burke/swish`

Semantic Web Toolkits

A quick look at Swish

Homepage:

<http://www.ninebynine.org/RDFNotes/Swish/Intro.html>

- Haskell framework for performing deductions on RDF data
- Graph isomorphism equality and differences
- Inference engine
 - ▶ Forward chain reasoning
 - ▶ Backward chain reasoning
 - ▶ Proof checking module
- Horn-style rule implementations
- RDF formal semantics entailment rule implementation
- Ready-to-run command line and script-driven programs

Haskell libraries

Looking at hsparql

- **Author** Jeff Wheeler
- **Maintainer** Rob Stewart
- **Source code**

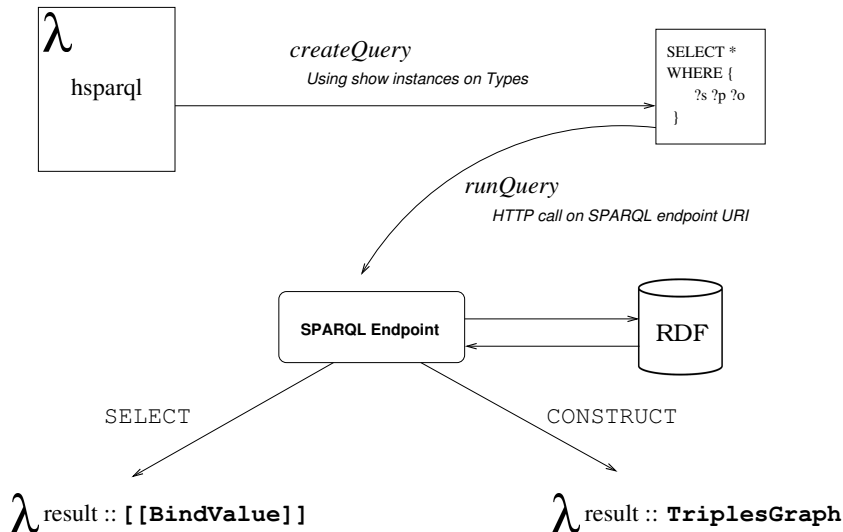
- ▶ `git clone git://github.com/robstewart57/hsparql.git`

hsparql includes a SPARQL domain specific language...

- Programmatically create SPARQL queries
- Includes functions for submitting SPARQL queries
 - ▶ For CONSTRUCT, SELECT, DESCRIBE and ASK queries
- Includes SPARQL response data structures
 - ▶ SPARQL Query Solutions (for select and ask queries)
 - ▶ RDF graphs (for describe and construct queries)
 - ★ Adopting the types from the 'rdf4h' package

Hsparql

Flow



Hsparql

Generating SPARQL Queries - Types

```
data Duplicates = NoLimits | Distinct | Reduced
```

```
data Prefix = Prefix String IRIRef
```

```
data IRIRef = IRIRef String  
            | PrefixedName Prefix String
```

```
data RDFLiteral = RDFLiteral String  
                | RDFLiteralLang String String  
                | RDFLiteralIRIRef String IRIRef
```

```
data Relation = Equal | NotEqual | LessThan | GreaterThan  
              | LessThanOrEqual | GreaterThanOrEqual
```

```
data Operation = Add | Subtract | Multiply | Divide
```

```
data OrderBy = Asc Expr  
             | Desc Expr
```

Hsparql

Generating SPARQL Queries - Types

```
-- |Permit variables and values to seamlessly be put into 'triple'
-- class TermLike a where

data QueryType = SelectType | ConstructType | TypeNotSet

data QueryForm = SelectForm QueryData | ConstructForm QueryData

-- |Local representations of incoming XML results.
data BindingValue = Bound Data.RDF.Node    -- ^RDF Node (UNode, BNode, LNode)
                  | Unbound                -- ^Unbound result value
                  deriving (Show, Eq)

data ConstructQuery = ConstructQuery
  { queryConstructs :: [Pattern] }

data SelectQuery = SelectQuery
  { queryVars :: [Variable] }
```

Hsparql

Examples of show instances

```
instance QueryShow Relation where
  qshow Equal      = "="
  qshow NotEqual   = "!="
  qshow LessThan   = "<"
  qshow GreaterThan = ">"
  qshow LessThanOrEqual = "≤"
  qshow GreaterThanOrEqual = "≥"
```

```
instance QueryShow OrderBy where
  qshow (Asc e)  = "ASC(" ++ qshow e ++ ")"
  qshow (Desc e) = "DESC(" ++ qshow e ++ ")"
```

```
instance QueryShow QueryForm where
  qshow (SelectForm qd) = unwords
    [ "SELECT"
    , qshow (duplicates qd)
    , qshow (vars qd)
    ]
```

```
qshow (ConstructForm qd) = "CONSTRUCT { " ++ qshow (constructTriples qd) ++ " }"
```

Hsparql

Generating a simple SELECT query

```
PREFIX resource: <http://dbpedia.org/resource/>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?page
WHERE {
    ?x dbprop:genre resource:Web_browser .
    ?x foaf:name ?name .
    ?x foaf:page ?page .
}
```

Hsparql

Generating a simple SELECT query

```
simpleSelect :: Query SelectQuery
simpleSelect = do
  resource <- prefix "dbprop" (iriRef "http://dbpedia.org/resource/")
  dbpprop  <- prefix "dbpedia" (iriRef "http://dbpedia.org/property/")
  foaf     <- prefix "foaf" (iriRef "http://xmlns.com/foaf/0.1/")

  x      <- var
  name   <- var
  page   <- var

  triple x (dbpprop ... "genre") (resource ... "Web_browser")

  triple x (foaf ... "name") name
  triple x (foaf ... "page") page

  return SelectQuery { queryVars = [name, page] }
```


Hsparql

Generating a simple CONSTRUCT query

```
simpleConstruct :: Query ConstructQuery
simpleConstruct = do
    resource <- prefix "dbpedia" (iriRef "http://dbpedia.org/resource/")
    dbpprop   <- prefix "dbprop"  (iriRef "http://dbpedia.org/property/")
    foaf      <- prefix "foaf"    (iriRef "http://xmlns.com/foaf/0.1/")
    example   <- prefix "example" (iriRef "http://www.example.com/")

    x      <- var
    name   <- var
    page   <- var

    construct <- constructTriple x (example ... "hasName") name

    triple x (dbpprop ... "genre") (resource ... "Web_browser")
    triple x (foaf ... "name") name
    triple x (foaf ... "page") page

    return ConstructQuery { queryConstructs = [construct] }
```

Hsparql Under The Hood

Submitting SPARQL SELECT query

```
-- |Transform the 'String' result from the HTTP request into a two-dimensional
-- table storing the bindings for each variable in each row.
structureContent :: String -> Maybe [[BindingValue]]

-- |Connect to remote 'EndPoint' and find all possible bindings for the
-- 'Variable's in the 'SelectQuery' action.
selectQuery :: EndPoint -> Query SelectQuery -> IO (Maybe [[BindingValue]])
selectQuery ep q = do
    let uri      = ep ++ "?" ++ urlEncodeVars [("query", createSelectQuery q)]
        request  = replaceHeader HdrUserAgent "hsparql-client" (getRequest uri)
        response <- simpleHTTP request >>= getResponseBody
    return (structureContent response)
```

Hsparql Under The Hood

Submitting SPARQL CONSTRUCT query

```
-- |Connect to remote 'EndPoint' and construct 'TriplesGraph' from given
-- 'ConstructQuery' action. /Provisional implementation/.
constructQuery :: forall rdf. (RDF rdf) =>
  Database.HSparql.Connection.EndPoint -> Query ConstructQuery -> IO rdf
constructQuery ep q = do
  let uri      = ep ++ "?" ++ urlEncodeVars [("query", createConstructQuery q)]
  rdfGraph <- httpCallForRdf uri
  case rdfGraph of
    Left e -> error $ show e
    Right graph -> return graph

httpCallForRdf :: RDF rdf => String -> IO (Either ParseFailure rdf)
httpCallForRdf uri = do
  let h1 = mkHeader HdrUserAgent "hsparql-client"
      h2 = mkHeader HdrAccept "text/turtle"
      request = Request  rqURI = fromJust $ parseURI uri
                        , rqHeaders = [h1,h2]
                        , rqMethod = GET
                        , rqBody = ""

  response <- simpleHTTP request >>= getResponseBody
  let content = E.decodeUtf8 (B.pack response)
  return $ parseString (TurtleParser Nothing Nothing) content
```

Hsparql

Hsparql in use

```
selectExample :: IO ()
selectExample = do
    (Just s) <- selectQuery "http://dbpedia.org/sparql" simpleSelect
    putStrLn . take 500 . show $ s

askExample :: IO ()
askExample = do
    res <- askQuery "http://dbpedia.org/sparql" simpleAsk
    putStrLn $ "result: " ++ show (res::Bool)

describeExample :: IO ()
describeExample = do
    rdfGraph <- describeQuery "http://dbpedia.org/sparql" simpleDescribe
    mapM_ print (triplesOf rdfGraph)

constructExample :: IO ()
constructExample = do
    rdfGraph <- constructQuery "http://dbpedia.org/sparql" simpleConstruct
    mapM_ print (triplesOf rdfGraph)
```

Hsparql

A more interesting SELECT example...

```
berliners :: Query SelectQuery
berliners = do
  xsd <- prefix "xsd" (iriRef "http://www.w3.org/2001/XMLSchema#")
  prop <- prefix "prop" (iriRef "http://dbpedia.org/property/")
  dbo <- prefix "dbo" (iriRef "http://dbpedia.org/ontology/")
  foaf <- prefix "foaf" (iriRef "http://xmlns.com/foaf/0.1/")
  resc <- prefix "resc" (iriRef "http://dbpedia.org/resource/")

  name <- var
  birth <- var
  death <- var
  person <- var
  knownfor <- var

  triple person (prop .. "birthPlace") (resc .. "Berlin")
  triple person (dbo .. "birthdate") birth
  triple person (foaf .. "name") name
  triple person (dbo .. "deathdate") death

  filterExpr $ birth <. ("1900-01-01", xsd .. "date")

  optional $ triple person (prop .. "KnownFor") knownfor

  return SelectQuery { queryVars = [name, birth, death, person, knownfor] }
```

Hsparql

A more interesting SELECT example *result*

```
[
  [LangLiteral "Wilhelm II" "en",
    TypedLiteral "1859-01-27" "http://www.w3.org/2001/XMLSchema#date",
    TypedLiteral "1941-06-04" "http://www.w3.org/2001/XMLSchema#date",
    URI "http://dbpedia.org/resource/Wilhelm_II,_German_Emperor",
    Unbound]
, [LangLiteral "German Emperor Wilhelm II" "en",
    TypedLiteral "1859-01-27" "http://www.w3.org/2001/XMLSchema#date",
    TypedLiteral "1941-06-04" "http://www.w3.org/2001/XMLSchema#date",
    URI "http://dbpedia.org/resource/Wilhelm_II,_German_Emperor",
    Unbound]
, [LangLiteral "Edmund Landau" "en",
    TypedLiteral "1877-02-14" "http://www.w3.org/2001/XMLSchema#date",
    TypedLiteral "1938-02-19" "http://www.w3.org/2001/XMLSchema#date",
    URI "http://dbpedia.org/resource/Edmund_Landau",
    Unbound]
]
```

Hsparql

Another interesting SELECT example...

```
selectExample :: IO ()
selectExample = do
  (Just s) <- selectQuery "http://api.talis.com/stores/bbc-backstage/services/sparql" drWhoQuery
  putStrLn . show $ s

drWhoQuery :: Query SelectQuery
drWhoQuery = do
  rdfs <- prefix "rdfs" (iriRef "http://www.w3.org/2000/01/rdf-schema#")
  po   <- prefix "po"  (iriRef "http://purl.org/ontology/po/")
  dc   <- prefix "dc"  (iriRef "http://purl.org/dc/elements/1.1/")

  position <- var
  title    <- var
  syn      <- var
  series   <- var
  episode  <- var

  triple (iriRef "http://www.bbc.co.uk/programmes/b006q2x0#programme") (po ... "series") series
  triple series (dc ... "title") "Series 3"
  triple series (po ... "episode") episode

  triple episode (dc ... "title") title
  triple episode (po ... "position") position
  triple episode (po ... "short_synopsis") syn

  return SelectQuery { queryVars = [position, title, syn] }
```

Hsparql

Dr Who results

Position	Title	Syn
6	The Lazarus Experiment	Martha has to save her family from the schemes of the monstrous Professor Lazarus.
13	Last of the Time Lords	Earth has been conquered and the Master rules supreme. Can Martha Jones save the world?
2	The Shakespeare Code	In Elizabethan England, William Shakespeare is under the control of witch-like creatures.
10	Blink	Only the Doctor can stop the Weeping Angels, but he's lost in time.
5	Evolution of the Daleks	As a new Dalek Empire rises in 1930s New York, the Doctor must enter an unholy alliance.
4	Daleks in Manhattan	The Doctor finds his oldest enemies at work on top of the Empire State Building.
7	42	The Doctor and Martha have to stop a spaceship from hurtling into the sun.
8	Human Nature	A schoolteacher called John Smith dreams of adventures in time and space.
11	Utopia	Captain Jack Harkness storms back into the Doctor's life.
12	The Sound of Drums	Harry Saxon becomes Prime Minister, but his dark ambitions reach beyond the stars.
1	Smith and Jones	When Martha Jones finds herself on the moon, she meets a stranger called the Doctor.
3	Gridlock	The Doctor takes Martha to New Earth, only to find that the city has become a deadly trap.
9	The Family of Blood	It's 1913 and war comes to England early, as the terrifying Family hunt for the Doctor.

A Plan?

Towards a Semantic Web Toolkit for Haskell

Future Work

- rdf4h

- ▶ Move towards feature parity with Jena Models

- swish

- ▶ Complete UTF-8 handling
- ▶ Internals: move to *polyparse*, *LookupMap*...

- hsparql

```
askQuery      :: EndPoint -> Query AskQuery -> IO Bool
describeQuery :: EndPoint -> Query DescribeQuery -> IO TriplesGraph
```

- A unified semantic web Type representation!

- ▶ Have all 3 packages use the same representations
 - ★ *prefix mappings, URIs, typed literals*...
- ▶ Unified serialisation of XML, N3, Turtle

Towards a Semantic Web Toolkit

A place for Haskell in the Semantic Web

- A feature rich RDF library
 - ▶ Providing an array of high level APIs working with RDF
- and**
- SPARQL construction & execution
 - ▶ *Programmatically* generate and execute SPARQL queries...
 - ▶ Living within the Haskell type system along the way
- and**
- A scripting interface to IO with RDF
 - ▶ To read, write & merge RDF graphs
 - ▶ Apply forward and backward chain reasoning rule inference
 - ▶ Compare graphs for equality

The end.